MySTEMGrowth Application

Design Document

sdmay26-27
Client/Instructor: Dr. Rover,
Consultant: Yiqi Lang
Caleb Hemmestad: Full Stack Developer
Ethan Buenting: Backend/Cloud
Ethan Van Caster: Frontend Developer
Nina Gadelha: Cybersecurity Developer
Ryan Mamrot: Frontend Developer
Sam Craft: Backend Developer/Testing
sdmay26-27@iastate.edu
https://sdmay26-27.sd.ece.iastate.edu/

Executive Summary

MySTEMGrowth Survey Tool is a web-based application that is designed in order to assess and provide visual results regarding undergraduate student's growth during participation in STEM research programs. Many existing programs rely on disconnected survey tools or manual data collection methods, making it difficult for students to reflect on their growth/development throughout the course. These types of tools can also make it challenging for staff to effectively evaluate the outcomes. This problem is important to solve as measuring student growth is critical for improving program quality, securing resources or funding and ensuring meaningful educational experiences.

In order to address this need, the system was designed with several key requirements, including but not limited to; protection of sensitive data, scalability to support different device sizes, improved cloud service management, survey recalculation/restructure, implementation of middleware and overall improvement on the user experience. In addition to these features, the application needed to support multiple user roles; students, program coordinators and administrators. Each role has distinct permissions and access to different interfaces in order to execute their expected duties. Students must be able to join a course, complete their survey and view their results and coordinators need to distribute surveys and manage participants. Administrators require full system oversight, while delegating coordinators and collecting results across multiple courses.

Our system follows a modern full-stack web architecture. The frontend is built using React and Next.js in order to create a responsive and intuitive user interface, while the backend was created by using Node.js and Express to handle application logic and API endpoints. MySQL database is used for data storage, enabling efficient retrieval of data and the comparison of survey results. Our cloud was switched from AWS to DigitalOcean in order to reduce cost while still guaranteeing scalability and reliability. Finally, security upgrades were implemented, through increasing password security, token-based authentication and role-based access control.

Our design effectively meets the requirements by providing a centralized, user-friendly platform that simplifies both data collection and data analysis. The application functionality has been validated through unit, integration and system testing. This ensures that core functionality, such as authentication, survey completion and role-based access perform as expected. Testing reveals what functionality is not only working correctly but also allows us to collect feedback from intended users and our clients. From there, we can make changes that better align with the received feedback, in order to continue improving the application. So far, testing and feedback has revealed our system improvements have successfully improved interface usability and reduced security risks.

The MySTEMGrowth Survey Tool has several important implications for both educational institutions and research programs. Providing a centralized, data-driven approach for measuring student growth enables students to better understand/reflect on their experience and growth throughout their course. Results can also be used as feedback for coordinators or instructors, allowing them to provide mentorship strategies and allocate certain resources. In addition, MySTEMGrowth is partnered with researchers from the University of Iowa in order to better analyze results from student responses. These results could potentially be used later on in order to improve certain aspects of the undergraduate research programs.

Learning Summary

Development Standards & Practices Used

Throughout the MySTEMGrowth project, our team applied software engineering practices to ensure reliability, security, and maintainability. We followed a structured software life-cycle model aligning with IEEE 1448a-1996, working though through analysis, design, implementation, testing, and deployment phases. Configuration management practices based on IEEE 828-2012 helped guide our use of Git for version control, feature branching, issue tracking, and controlled merges. Verification and validation principles from IEEE 1012-2016 helped navigate our testing strategy, including unit, integration, system, and acceptance testing.

Summary of Requirements

- A web-based tool that client users can access
- Create charts based on survey data
- Role-based access for administrators, program coordinators, and students
- Secure account creation, login, logout, and account management
- Ability for students to take surveys and view personal results
- Ability for coordinators to view aggregated and student results within assigned groups
- Administrative control over surveys, programs, users, and exports
- Secure storage of user data and survey results
- Responsive UI supporting desktop and mobile devices
- Cloud-hosted deployment with scalable backend and managed database
- Automated build and deployment pipeline
- Data export and reporting functionality

Applicable Courses from Iowa State University Curriculum

- COM S 2270
- COM S 2280
- SE 3090
- SE 3190
- COM S 3630
- SE 3170
- SE 3390
- CPR E 2300
- CPR E 2310
- CPR E 3310
- ENGL 3140

New Skills/Knowledge acquired that was not taught in courses

- Database management
- App deployment
- Cloud services
- Configuring and migrating cloud
- Cookies and token-based authorization
- Implementing middleware
- RBAC across frontend and backend
- Some forms of testing

Table of Contents

List of figures/tables/symbols/definitions

List of Figures:

List of Tables:

List of Symbols and Definitions:

API - Application Programming Interface

CI/CD - Continuous Integration / Continuous Deployment

DB - Database

JWT - JSON Web Token

RBAC - Role-Based Access Control

UI - User Interface

UX - User Experience

1. Introduction

1.1. Problem Statement

The MySTEMGrowth Survey Tool was developed to address the challenge of effectively measuring and analyzing undergraduate student growth during participation in STEM research programs. This project was inherited from 2 previous Senior Design teams. These teams laid the groundwork and designed the foundation for the application. Although some functionality was implemented, our clients expressed key features that were missing and some components that needed to be created or improved upon.

In this tool's previous implementation, as well as many other existing programs, limited insight regarding how students develop key skills was provided. The lack of clear and accurate evaluation regarding the old tool made it difficult to analyze the survey results. On top of that, there were some issues with the survey itself, which resulted in inaccurate measurements. Our main goals when innovating the survey tool was to improve the survey itself and the display of results, provide a better user experience by upgrading the user interface, modifying the scalability to allow different device sizes to access the application and ensuring professional programming practices were implemented.

During the beginning of their research course, students will take the survey, but creating an account, joining a specific course and taking the survey. Their first set of results will give students a baseline. After some time in their course, they will be asked to take the survey again. They can then compare these results with their first set of results to see what areas of STEM they have improved on. Depending on the course/faculty member, the students will be expected to take the survey multiple times, each time comparing their current results with the most recent completion of the survey. Results are used by students to analyze what they have learned in the course and how it has contributed to their growth. Program coordinators and administrators are also able to view results, to understand what areas of STEM their students are improving on. These results are also sent to the University of Iowa, where researchers are analyzing the anonymous student data in order to aid them in their research.

The MySTEMGrowth Survey Tool is used across multiple schools/universities across the midwest. We wanted to ensure this tool was as accurate and user friendly as possible, as many students will be expected to navigate the application while completing undergraduate research courses. We wanted to ensure participants were able to easily access the survey, view results in a clear manner and also compare their results when retaking the survey throughout the course. We believe it is crucial for students to clearly understand their progress in order to get as much out of the research course as possible. In addition, it is also important for faculty to observe the survey results of students so they can potentially add helpful resources or potentially tailor aspects of the course in order to help students improve upon a certain area of growth.

1.2. Intended Users

The MySTEMGrowth Survey Tools was designed to support 4 different types of users

1) Participants/Students
2) Program Coordinators
3) Administrators
4) Researchers

Each user group will have different roles, permissions and interfaces that allow them to carry out their expected responsibilities.

1. Participants

Participants have the lowest level of access/permissions out of all the roles, as they are only allowed to view information/data that is their own. Their user interface is straightforward and follows a chronological-like path, as they have limited access to capabilities. Participants will be able to view their personal data at any time, but will not have access to anyone's information that is not their own. Participants roles include:

- Creating an account
- Joining their authorized program
- Taking the survey
- Viewing/comparing their results
- Edit their account information
- Accessing resources

2. Program Coordinators

Program Coordinators will have more access than a specific participant, as they will be managing the course itself, and therefore, all students in the program. Their interface is also designed to be simple to use and detailed, in order to differentiate their programs, if they are running multiple. Program Coordinators roles include:

- Creating an account
- View programs
    - Delegate surveys
    - Add/remove students from their program
    - Request demographic information
    - Export participant results (survey + graph)
    - Manage student survey progress
- Add resources for specific programs
- Edit their account information
- Generate/distribute program codes

3. Administrators

The Administrator role oversees all processes within the survey tool. Like the Program Coordinator, they can manage aspects about a specific program or student. They also have the responsibility of delegating Program Coordinator as well as exporting results of a specific student or

program in order to transfer them to the researchers at the University of Iowa. Administrator roles include:

- Creating an account
- Assign Program Coordinator roles
- Modify survey
- Inviting/removing participants (from any program(s))
- Viewing results for any participant (any program)
- Exporting data/sending results
- Edit their account information
- Deleting program(s)

4. Researchers

The researcher role only has one duty and that is to be able to anonymously export data. We created this role in order to give researchers a way to automatically collect the data anonymously, rather than having to collect the data through the administrator role (which does not have access to anonymous results). This eliminates an extra step the researchers had to make in order to use the data for their studies, as it needed to be anonymous. Researcher roles include:

- Creating an account
- Edit their account information
- Export data (all programs, all students)

2. Requirements, Constraints, and Standards

2.1. Requirements & Constraints

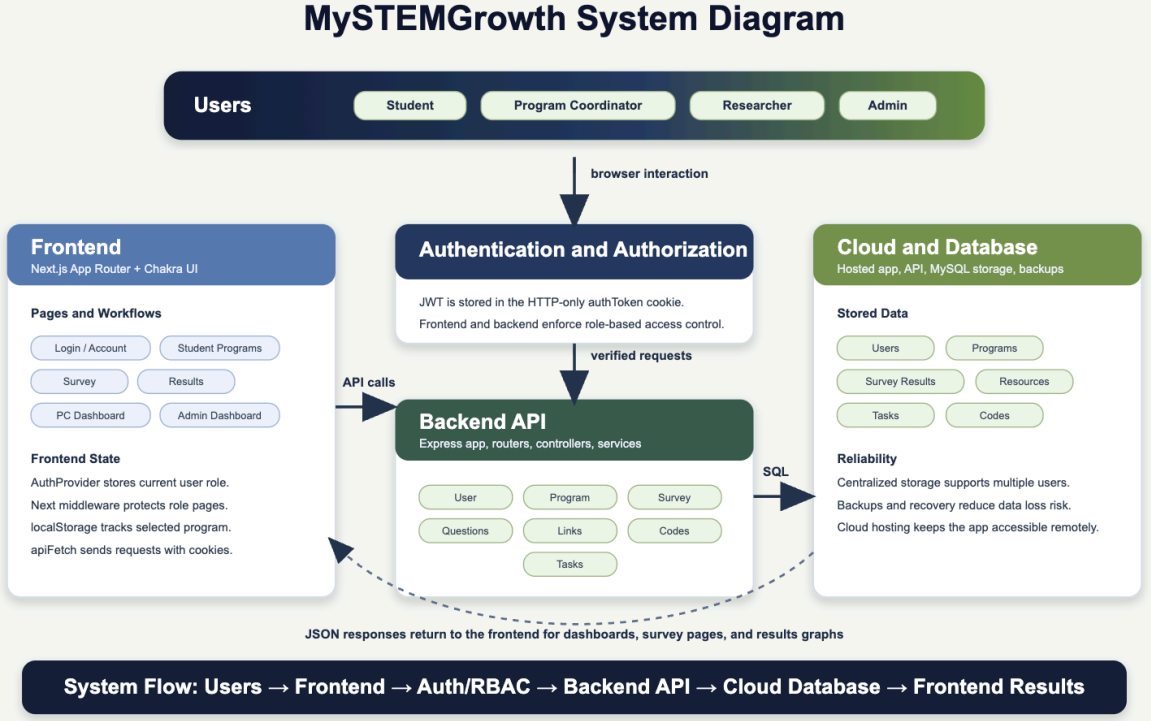


Figure 1: System Diagram

2.1.1 Functional Requirements

- Participants should be able to
    - Join a program
    - Take a survey (when appointed to them by a program coordinator)
    - View results (view in website and download)
    - Compare results after taking the survey 2+ times
    - Access resources
- All users (administrators, program coordinators and students) will be able to create an account, login to their account, view survey results and edit their account information
    - Students should only be able to view their own survey results (from any time period)
    - Program Coordinator should have access to view any/all results from students within groups they have assigned
    - Administrators should be able to view all survey results (any student from any program)
- Administrators will be able to modify the survey (add, edit or delete questions), have access to all results and enable program coordinators to create and distribute surveys to approved groups

2.1.2 Resource requirements

- Frontend
    - React.js
    - Next.js
    - Chakra UI
- Backend
    - Node.js
    - Express Framework
    - Next Auth
    - MySQL
    - PGAdmin
    - Jest
- Cloud
    - Digital Ocean
        - Managed Database
        - App Platform encrypted env vars
            - Use Web Services to function as components
        - DNS

2.1.3 UI requirements

- Each user type page should have different options based on granted permissions
- A universal Navbar should be present on every page (regardless of user type or specific page)
    - Navbar can have different options/buttons based on user access
- Login page will enable user to login or create a new account
- Change in theme (colors, images/icons, displayed text/information)
- Innovation to actual survey
    - Fix previous errors (repeated questions, grammatical errors, etc)
    - Fix formatting (same number of questions per page)
- Add authentication for administrators
    - Implemented cookies/tokens
- Innovate survey results page
    - Add numbers to results displayed
    - Change display type (edit display circles)
- If slider bar is used to answer questions, set default to 50%
- After completing a page, ensure next page starts user at the beginning of the next page
- Save progress if survey was not completed in one sitting (up to 1 week)
- Allow program coordinators to view what students (in their groups) have completed the survey
- Allow tool to be viewed and taken on mobile devices (add modularity)
- Improve overall security of the website

2.2. Engineering Standards

Engineering standards are important for two primary reasons. First is that they uphold safety and privacy standards for the general public. Secondly, they uphold a standard for both software and hardware to be compatible with other devices of different manufacturers, and uphold quality standards while doing so. Engineering standards allow engineers to design systems that are interoperable between different devices, are safe for the consumer, and can be trusted by the general public.

IEEE 1448a - 1996: Standard for Information Technology: Software Life Cycle Processes

This standard ensures that our team has a specific plan to execute. It defines clear steps and guidelines for understanding, planning, developing and executing when innovating the MySTEMGrowth Survey tool. IEEE 1448a adds compliance methods and clarifications in order to make the cyclical process more adoptable to company practices which provides guidelines on development and software practices. The overarching goal is to provide stability, flow, consistency and organization in software projects.

IEEE 828-2012: Standard for Configuration Management in Systems and Software Engineering

The IEEE 828-2012 Standard defines processes and requirements for configuration management over the life cycles of a software application. It is a set of processes that provide guidelines for tracking and documenting changes to different parts of the program, such as source code, documentation, and databases. The guidelines set out a particular set of rules that require that changes are properly documented and implemented in a controlled manner. This standard helps to support practices such as version control and change tracking.

IEEE 1012-2016: Standard for System, Software, and Hardware Verification and Validation

This standard was created to provide guidelines for the verification and testing of different system applications. It helps to define what product verification looks like, and how to verify that it meets the intended purpose of development. The standard promotes performing multiple levels of testing to ensure that the system meets its intended requirements. It also promotes documentation and risk management to ensure that the application meets its pre-defined requirements and is reliable.

Project Relevance to IEEE Requirements

IEEE 1448a - 1996 (Standard for Information Technology: Software Life Cycle Processes) is very relevant to our project. Working in cycles will allow us to focus on a common goal and ensure everyone is on the same page about the plan moving forward. It ensures that one member won't rush ahead or fall behind, allowing everyone to participate in teamwork more effectively. It also will help us stay more organized and allows us to better retrace our steps in case of any issues.

IEEE 828-2012 is relevant to our project because it helps us to create a set of guidelines for our transition of cloud hosting from Amazon Web Services to Digital Ocean. We currently have a function application to begin with, but we are switching providers due to cost and simplicity of the hosting services. However, we want to have little to no data loss during the transition period with as little downtime as possible. Using this standard to create a set of guidelines, our team will be able to make the transition smoothly and document our progress along the way.

IEEE 1012-2016 will be used by our team to perform additional verification on the existing portion of the application, as well as the new features we implement throughout the next two semesters. While we won't be able to test for all possible scenarios, we will be able to place the application through a wide variety of tests in order to ensure that our product meets the expectations and intended usage as determined by our senior design team along with our advisor.

Additional IEEE Standards

Another IEEE standard our group thought about incorporating was IEEE 29148-2018. This standard is all about requirements engineering, which is relevant to our project due to our goal of adding additional features to the application. While this is a relevant standard, we felt that the previous three standards better incorporated our goals going forward with this project.

Project Modifications for Adherence

In order to adhere to standards that are demonstrated by IEEE 1448a - 1996 (Standard for Information Technology: Software Life Cycle Processes), our team will be completing this project in phases. These phases (currently) include: analysis, requirements, brainstorming/design, development, testing/maintenance and deployment. Stepping through this structure allows us to have a common understanding of the phase/stage we are currently working on. Although everyone will have their own responsibilities, we will be working on the same concept, so we can discuss, help each other and test features on a "scheduled timeline".

Adherence to the IEEE 828-2012 standard will not require many changes to the project, but more importantly the documentation and creation of guidelines for our transition of cloud hosting services. In order to adhere to the IEEE 1012-2016 standard, our group will be administering more forms of testing for the application, including both tests by our team and by actual undergraduate students. It is through adherence to this standard that we aim to develop a ready-to-release application by the end of our project timeline.

3. Project Plan

3.1. Project Management/Tracking Procedures

Our team chose to adopt a mix of both agile and waterfall project management styles throughout 4910 and 4920. Overall, we believed that the waterfall approach would be the most beneficial, when talking about completing the project as a whole. Communicating with our clients in order to identify all requirements/improvements they wanted us to implement gave us a clear vision for planning, designing and eventually, implementing. Ensuring the design of the project is clearly laid out and approved by our clients ensured

This semester, our team has utilized the 'issue boards' feature available on Git in order to track progress. This feature allows us to create features we intend to add to the current state of the survey. We are able to label/categorize issues to ensure they get implemented in all required parts of the project (frontend, backend, cloud). We are also able to claim issues so we know who is responsible for what features and add timeline goals with dates. We can further organize details by labeling them with specific categories; such as security, UI, database, etc. Overall, the Git issue

boards are an easy method we can use to ensure we are making progress in a timely manner and keep everyone organized and accountable.

We also used our team meetings to discuss progress and demonstrate contributions. Towards the end of the semester, we also started creating excel tracker sheets in order to better communicate to our clients what progress was being made by team members. We essentially took the information from the Git issue boards and transferred it into the excel sheets (since our clients do not have access to our Git Project).

3.2. Task Decomposition

All features/tasks we implemented in the MySTEMGrowth Survey Tool fell under one of the following four categories; Frontend, Backend, Security and Cloud.

- Frontend
    - Added support to use website on mobile devices
    - Changed UI for all roles
    - Update navbar
    - Improve PC UI to allow for clarity/more access to information
    - Allow PC to view progress of student's surveys
    - Resource library for quick and easy resource options
    - Edit account feature
    - Create a student dashboard to support students joining multiple programs
    - Partial survey saving
    - Incomplete question indicator and navigation
    - Added dropdown to select the data displayed on results page
    - Added option to display floating point score on results page
    - Created Researcher role
    - Updated survey data export to align with research anonymity requirements
- Backend
    - Update survey calculations to use floating point math
    - Save survey progress for partial completion
    - Fix errors within the survey
    - Multiple program support
    - Comparison for all results between programs
- Security
    - Add middleware
    - Implement cookies and tokens
    - Enable stronger password requirements
    - Protect endpoints
    - Password hashing
- Cloud
    - Migrated previous database from AWS to Digital Ocean
    - Changed DNS to reflect cloud migration
    - Fix CI/CD pipeline

3.3. Project Proposed Milestones, Metrics, and Evaluation Criteria

Milestone 1: Implement authentication with cookies and tokens

In order to improve security and overall application completion, we implemented middles, cookies and tokens. These three components work together in order to provide a secure and efficient way to verify user identity across the application.

When a user logs into their account, the server generates an authentication token that is used to uniquely identify the user, eliminating the need for the user to repeatedly enter their credentials. Middleware ensures that only authenticated users can interact with parts of the application, ensuring only authorized users get access to sensitive data.

Progress for this milestone is measured using quantifiable metrics such as login success rate, token validation accuracy and the percentage of protected routes that correctly block unauthorized access. All target goals for the above metrics 95% reliability and functionality, while improving the overall security of the application.

Milestone 2: Enable survey compatibility with smaller devices (mobile)

To improve accessibility and overall flexibility regarding user experience, we optimized the survey interface for small devices such as tablets and mobile devices. This involved implementing responsive design techniques and adding flexible layout elements in order to tailor the size of components on the website, based on the device size being used by the user.

Progress for this milestone is measured using metrics such as mobile usability rate, screen compatibility across device sizes and completion rate of surveys on smaller devices. All target goals for the above metrics 95% functionality, accessibility and flexibility.

Milestone 3: Improving the survey

Not only did we fix previous errors within the survey, we also added uniformity and innovative features in order to provide clarity regarding results for the users. These improvements not only ensure correctness of survey results but also make the survey appear more uniform, professional and accurate.

Some simple survey errors we fixed involved; deleting duplicate questions, fixing typos and fixing grammatical errors. We also improved the layout of the survey by ensuring every page had (roughly) the same amount of questions presented in the same format. We fulfilled some simple requests from our clients by setting the slider default to 50% (previously 0%), added partial survey saving and a missing question indicator. Finally, we improved the results page by allowing students to be able to choose which survey results to compare, added an option to display numerical results and made the format of the graph more clear.

Progress for this milestone is measured using metrics such as survey completion rate, average time to complete the survey and overall user satisfaction feedback. All target goals for the above metrics 95% user satisfaction, completion time and completion rate.

Milestone 4: Testing

One of the final tasks we performed on the application was testing it. We conducted multiple different types of testing, in order to ensure complete functionality. We had been continuously testing as we added features, as we adopted the agile testing method. This means, for each new feature we added, we tested the functionality of the new feature as well as the functionality of the rest of the application, ensuring any new work we completed did not break any previously working features.

Each week during our advisor meetings, we would update them with progress we made and demoed to them our changes. This way, we were able to get real time feedback from them, make the recommended changes and represent the feature at the next meeting. This always ensured that we were on the same page as our clients, and we were delivering the results they expected.

Finally, during the last few weeks of the project, we conducted user testing. We allowed students who had used the previous version of the application to access our version and test its functionality. We created a survey that they took after using our new implementation, took their feedback and used that information to populate this design document, our IRP presentation and our poster.

Progress for this milestone is measured using metrics such as client feedback and overall user satisfaction feedback. All target goals for the above metrics 90% user satisfaction.

3.4. Project Timeline/Schedule

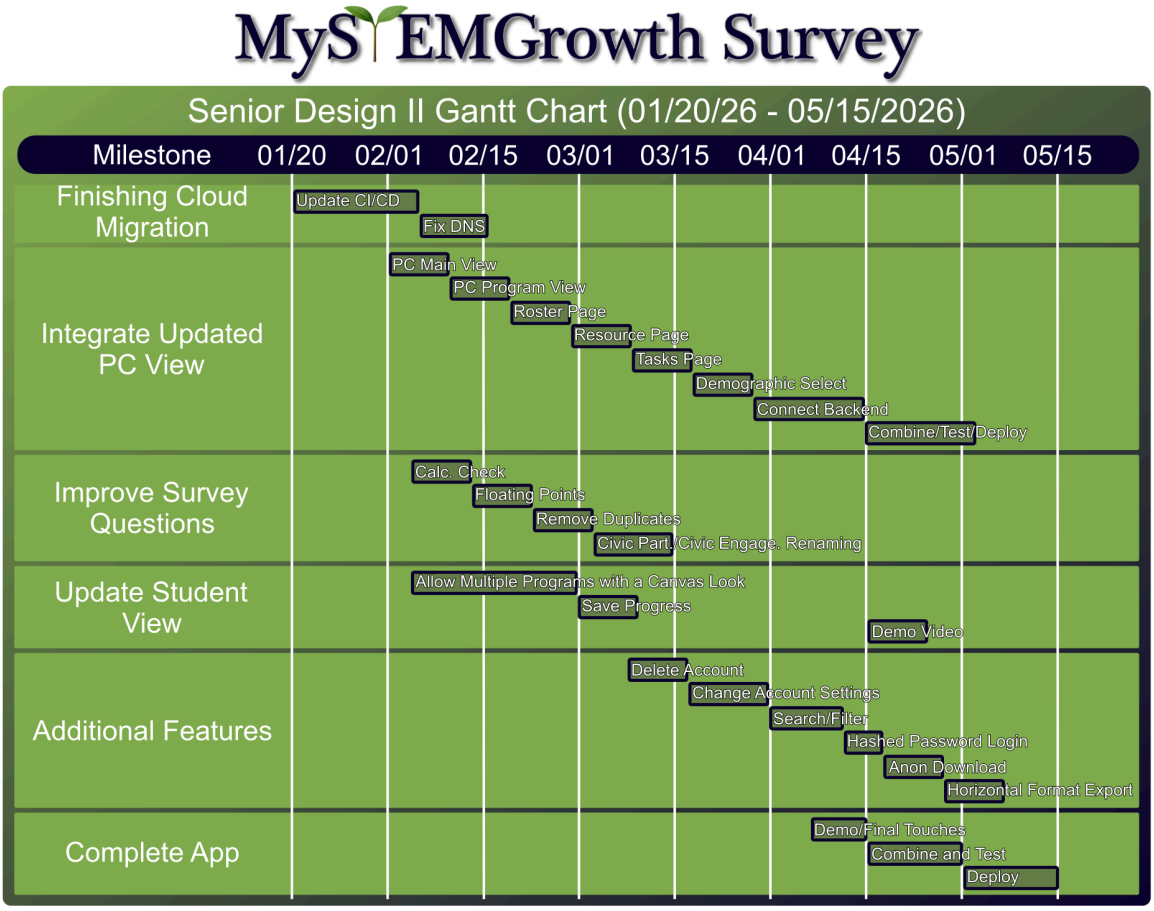


Figure 2: Gantt Chart

3.5. Risks and Risk Management/Mitigation

Risk 1: Cloud Migration

Migration from AWS's database to Digital Ocean was smooth as we faced very little issues. We ensured to keep all data/services active on the old database until we completed the migration and confirmed our new database would work with the application. This ensures no data would be lost and allows us to copy the format/structure of previous tables, reducing the time it took to copy the database. The most risky part we faced was ensuring we were able to get the old DNS back once we migrated, and we were able to do this no problem. In the previous iteration of the design document from CPRE 4910, we had this risk probability set at 25%.

Risk 2: Frontend/Backend Improvements

These changes were identified as low risks, only being about 5%. All changes were software based that were tested before pushing improvements to the production environment of the website. There were a few occasions when new innovations caused old functionality to break, however, we thoroughly tested features to ensure all functionality was restored. Since we used Git in order to

manage the website code and collaborate with each other, we also tried to use branches for different features, this way, if the website ever "broke" we could simply reverie back the working copy.

Risk 3: Survey Improvements

Although our initial plan involved integrating Qualtrics into our survey, in order to use their features, we decided against this due to the risk being high. Instead, we improved the survey created in the website itself, by editing the code to get rid of errors and improve the display/format of the survey as well as the results. Our custom implementation of the survey not only mitigated risks but also gave us more control over how we wanted the survey to look. Without using qualtrics, the risk of survey improvements was around 5%, as we could also revert changes and would never lose any data.

Risk 4: Security Improvements

Security improvements we made included; implementing middleware, tokens, cookies and password hashing. We also protected vulnerable endpoints and improved password security. When we initially adopted this app, we did vulnerability testing in order to determine what improvements could be made towards security. The previous version of the site had little to no security, so adding our own measures wasn't risky at all as we couldn't really make it any more vulnerable than it already was. There was potentially 5-10% risk as implementing any of these features incorrectly could add more exploit possibilities. However, we ensured testing on all features we added as well as reperforming our initial vulnerability test to ensure our features fixed previous issues.

3.6. Personnel Effort Requirements

Task	4910: Average time/week (per person)	4920: Average time/week (per person)
Switch Clouds (AWS → Digital Ocean)	3	2
Improve security	3	4
Modify survey (questions + format)	3	3
Improve UI (for all users)	4	5
Improve storage	2	1
Add new user features	3	3

- Switch Clouds
    - Migrate all current data
    - Ensure correct DNS
    - Delete old account/services
- Improve security
    - Hash + salt passwords

    - Add tokens/cookies
    - Protect endpoints
    - Add authentication
    - Add middleware
    - Perform vulnerability tests
- Modify Survey
    - Fix answer methods
    - Fix questions (duplications/misspelling)
    - If keeping sliders - default at 50%
    - Add save progress button
    - Auto scroll to top
    - Ensure correct calculations
    - Improve results display page
- Improve UI
    - Add modularity (correct format for small screens)
    - Change PC view from cards
    - Allow PCs to view who hasn't submitted survey (in managed groups)
    - Rework formatting for all views (students, PCs and admin)
    - Add info to Resources page
    - Add to navbar
    - Overall improve UI for all users (cleaner and more simple look)
- Improve storage
    - Delete unused fields
    - "Unsubscribe" from certain cloud options (saves money)
- Fix Github CI/CD pipeline
    - Write new CI/CD script
    - Add endpoints of Digital Ocean server
    - Test CI/CD script
    - Implement script for push requests in Gitlab

We found no substantial mismatches between estimates and actual effort. Improving the UI took more time than initially expected due to making more changes than we thought. Not only did we end up modifying the UI for all user roles, we also presented changes made to our clients, took their feedback and made changes based on that feedback, causing more time to be spent in this category. Cloud work also slowed down once the initial migration was completed. However, it took a lot of time at the start, so we think this overall evened out and took as much time as we anticipated.

3.7. Other Resource Requirements

Our project requires no additional physical resources.

4. Design

4.1. Design Context

4.1.1 Broader Context

Area	Description	Examples

Public health, safety, and welfare	Protects student data and promotes well-being by giving learners clear feedback on their research experience and growth. Security controls help prevent misuse of personal information.	Encrypted auth and least-privilege roles. Clear progress + "resume later" to lower survey fatigue. Admin tooling that avoids accidental exposure of student results.
Global, cultural, and social	Respects diverse backgrounds and programs by using inclusive language, accessible design, and flexible scales/question types. Supports different academic cultures and course structures.	Mobile-first layouts. Wording review to avoid cultural bias. Role views for admins, coordinators, and students.
Environmental	Digital-first delivery reduces paper and travel overhead; efficient cloud choices lower compute waste. Development practices consider energy usage of hosting and CI/CD.	Online surveys instead of printed packets. Adjusting the database size to match what we actually need. Static asset caching/CDN. Deleting stale accounts/data to reduce storage footprint.
Economic	Keeps costs sustainable for universities and research programs. Streamlines coordinator/admin time. Supports student success and program improvement that can attract funding.	Predictable hosting. Automated CI/CD to cut maintenance. Batch export of results for grant reports. low training/onboarding time due to simpler ops.

4.1.2 Prior Work/Solutions

One of the biggest advantages our application offers is the ability to collect survey data virtually, allowing for easier exportation for research purposes, or simply to make it easier to allow the student to view how their personal outlook and abilities have changed throughout their research experience. Using digital processes to collect information allows data to be processed more time efficiently, and has been used in a variety of different applications to help assess data and provide analytics [1].

The survey itself also holds similarity to previous projects that involved assessing undergraduate research program effects on students, such as the SURE survey. This survey was created to help collect student responses and evaluate things like student retention and educational development based on the survey responses, similar to one of the potential benefits listed for our project [2]. One of the primary actions they performed to solve issues was to ensure data comparability and standardization to ensure that results collected would be effective if program coordinators or researchers wanted to use the data later on.

The previous group, team sdmay25-24, has also had a lot of documentation and a solid code basis for us to work on and help us determine fixes and improvements we wanted to implement, as well as what worked well [3]. One of the biggest shortcomings we found was with the security of the application, as there was not much documentation, and there were notable security concerns with the application when we received it. However, there were many benefits to be had with having previous documentation as it made understanding the structure of the application very easy to understand and begin working on.

4.1.3 Technical Complexity

1. The design consists of multiple components/subsystems that each utilize distinct scientific, mathematical, or engineering principles

Our application consists of many different subsystems, all of which are listed and described here:

1.1. Frontend: This component uses React and Javascript to implement the user interfaces that our undergraduate users and other roles interact with. It is the primary tool that is used to collect data and display it in a clear and readable manner, whether that be for program coordinators to view how their program is influencing undergraduate students, or to allow students to see that they have grown according to their personal experience.

1.2. Backend: The backend is what allows our system to communicate with our databases and hosting environments, allowing for authentication and IO handling to create a seamless experience for users regardless of their role.

1.3. Database: The MySQL database management system allows for the application to store and sort data such as account information as well as survey questions and their response data.

1.4. Hosting: This application was previously implemented with AWS, but because of cost and ease of scalability we decided to migrate to Digital Ocean. Digital Ocean provided us with the exact same support application side at a fraction of the cost, and will be much easier to scale should the application grow in the future.

2. The problem scope contains multiple challenging requirements that match or exceed current solutions or industry standards.

2.1. Scalable Design: Our application requires the use of industry level design for the purpose of scalability. Our application needs to be able to grow with its usage, as it has the opportunity to be used in a number of research programs.

2.2. Data Privacy/Security: Our application now implements industry standard security features, such as hashing, the use of cookies/tokens, the protection of endpoints and the implementation of middleware to ensure that user data such as passwords and survey results are safe.

2.3. Cross-platform accessibility: While the previous application version functioned on a browser, our team worked on UI improvements to make the application easier to use, as well as offer a more readable view for admins or Program Coordinators to look over all responses for the respective programs that they oversee. In addition to that, we developed mobile support to allow users to take the survey or view their data on their mobile devices.

2.4. Automatic Deployment: We have implemented an automated CI/CD pipeline to allow our team, or any team following, to continue to develop the application and seamlessly integrate these changes to the web application without any interruption to service.

4.2. Design Exploration

4.2.1 Design Decisions

1. **Switching from AWS to Digital Ocean:**

Our team chose to migrate the MySTEMGrowth Survey's infrastructure from AWS to Digital Ocean after coming to an agreement that AWS was overly complex and costly for our project needs. Digital Ocean's App Platform and Managed Database services offered a simpler setup, predictable pricing, and an easy way to integrate with our existing Node.js backend and MySQL database. This transition has enabled for easier onboarding for future developers with less of a learning curve to get started. It also improved security through encrypted environment variables and managed backups while maintaining scalability and reliability. Overall, the move to Digital Ocean supports our goals of affordability, maintainability, and long-term sustainability for the MySTEMGrowth platform.

2. **Mobile layout support:**

One of the initial major requests from our clients was to add mobile support to the application, as it was not previously available. Our clients wanted to ensure the survey tool could be used on many different platforms, not just laptops. We added responsive design features that adjust the layout based on screen size. Some of these features include adding flexible components, scalable text, a modified taskbar, and mobile friendly navigation. By adding support for multiple size devices prioritizes accessibility and user reach. This overall improves flexibility, usability and scalability towards the application. We hope this will allow for app growth as it now can appeal to more users.

3. **Improve UI for all users:**

Although the previous implementation presented a UI that was functional, there were some features that it was lacking. Clarity and limited access to information were 2 big issues we wanted to address and improve upon. We made significant changes to all user UI screens in order to improve the experience for every type of user. Not only did we increase functionality by adding new features, but we overall improved navigation and styling of the application.

We wanted to keep a simple design in order to provide clarity to users, especially on the student side, as their permissions/access is limited. For the Program Coordinator and Administrator roles, we wanted to display more information regarding programs and students in order to provide clarity. We discussed and tested different types of layout options with our clients in order to ensure we designed a 'look' they were happy with.

4.2.2 Ideation

For at least one design decision, describe how you ideated or identified potential options (e.g., lotus blossom technique). Describe at least five options that you considered.

For the design decision of switching from AWS to Digital Ocean, our team used a collaborative brainstorming and comparison session to explore multiple hosting and cloud service options. We used a lotus blossom–style ideation approach, starting with the goal of finding a cloud

platform that meets our needs while reducing the cost and keeping the app easy to maintain and scalable in the future, then expanding outward with potential solutions and sub-ideas.

We initially identified five good options:

1. **Amazon Web Services (AWS)** – Continue using the existing infrastructure.
2. **Digital Ocean** – Migrate to a more developer-friendly and cost-effective environment.
3. **Google Cloud Platform (GCP)** – Leverage built-in integrations with CI/CD and analytics.
4. **Microsoft Azure** – Utilize advanced security and enterprise hosting capabilities.
5. **Heroku or Vercel** – Use a simplified app hosting service with auto-deploy pipelines.

After going through all the options, we eventually decided to go ahead with the switch to Digital Ocean. Since the migration, we have already seen payoffs. While keeping all the same functionality, we were able to reduce the cost from over $200 per month to $25 per month. This was one of the main motivations behind the switch, as we previously observed many unimplemented features with AWS that the application was being charged for. Since the MySTEMGrowth survey tool is not necessarily a highly trafficked application that is in use 24/7, we expected we would be able to switch to a different cloud host that offered less services, which would ultimately lead to reducing costs. Overall, we were able to save a substantial amount of money towards hosting the application without sacrificing any services/functionality.

4.2.3 Decision-Making and Trade-Offs

After looking at all five possible hosting options, our team evaluated each using a weighted-decision matrix based on five criteria: cost (25%), ease of setup (20%), integration with our stack (20%), security (20%), and scalability (15%). Each option was scored from 1 (poor) to 5 (excellent).

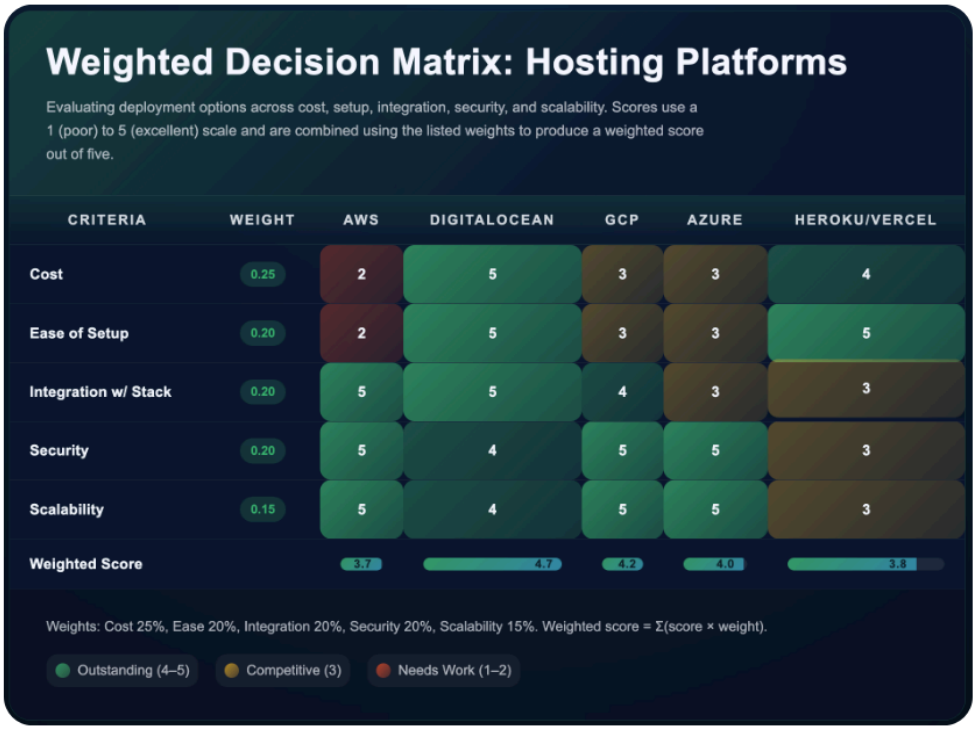


Figure 3: Decision Matrix

Digital Ocean achieved the highest overall score by producing most of the largest individual scores, and highest overall average out of the metrics we deemed most important. While AWS and Azure offered stronger enterprise-grade scalability, their pricing and configuration complexity made them less practical. Heroku and Vercel were intuitive but lacked control over databases and back-end customization.

The trade-off in choosing Digital Ocean was accepting slightly fewer enterprise integrations in exchange lower cost and less of a learning curve for future developers.. We believe this decision ultimately supports MySTEMGrowth's goals of maintainability, reliability, scalability, cost efficiency, and long-term sustainability.

4.3. Final Design

4.3.1 Overview

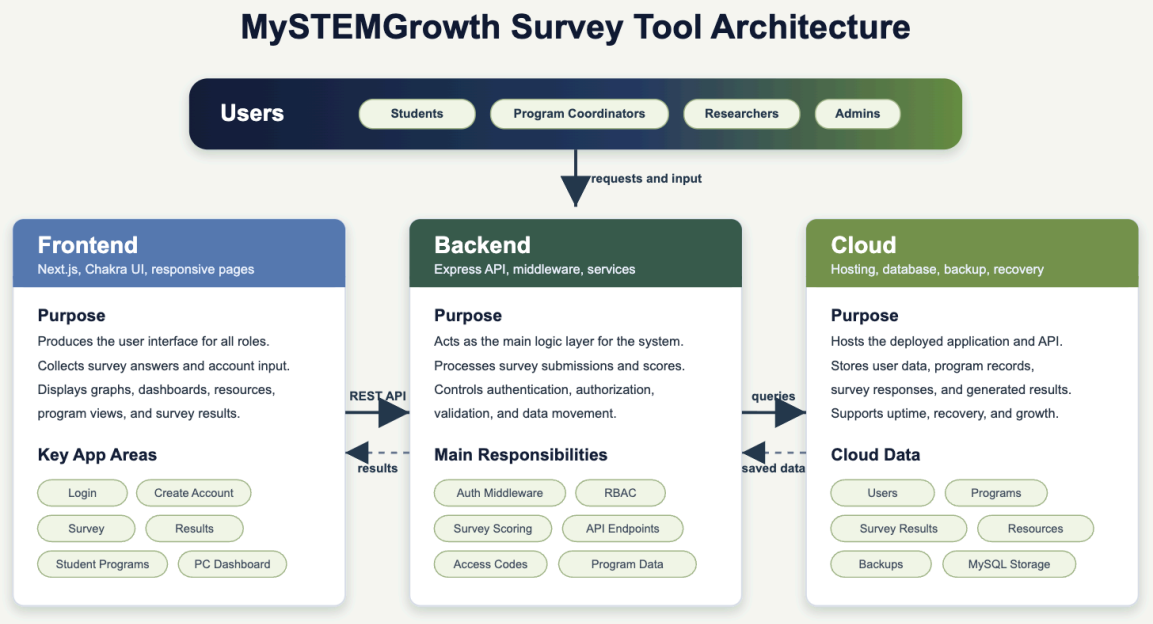


Figure 4: Tool Architecture

The MySTEMGrowth survey tool consists of 3 main components; cloud, frontend and backend. Each component serves a specific purpose, but all work together to provide functionality in order to support user interaction and engagement.

The cloud component is used to store data and host the application, allowing it to be accessed from anywhere. It allows user information and survey responses to be saved reliably and ensures the system can scale to support multiple users at once. In addition, the cloud supports data backup and recovery, helping prevent data loss and contributes to the system's overall reliability and uptime.

The frontend side of the application is responsible for producing the user interface that all user types interact with. This includes displaying different pages (login page, home page, survey page(s), etc), collecting responses when taking the survey, displaying survey results through a graph design and ensuring the application is flexible to use across multiple platforms, such as laptops, tablets and mobile phones.

Finally, the backend component of the system acts as the brain. It is responsible for handling logic, such as processing survey responses, calculating survey results, managing user authentication and controlling how data is moved between the user interface and storage. The main responsibility of the backend is to ensure that user data is handled correctly, securely and as intended.

4.3.2 Detailed Design and Visual(s)

MySTEMGrowth is a role-based web application (Student, Program Coordinator, Admin) delivered as a responsive React/Next.js client that talks to a Node.js/Express API, backed by a managed MySQL instance on DigitalOcean. Persistent assets (exports/logs) live in object storage. Authentication uses hashed passwords and session cookies/tokens; authorization is enforced via RBAC on every API boundary. CI/CD builds and deploys the app platform components on push. See Figure 1 (System Architecture) and Figure 2 (Components and Data Paths).

Frontend (React/Next.js)

- **Survey UI**: renders question types (slider/likert/text), validates inputs, supports "save progress."
- **Results Dashboard**: charts/filters for individuals and cohorts; CSV export.
- **Role Views**: conditional navigation and routes per Student/PC/Admin.
- **HTTP Client Layer**: fetch wrapper adds auth headers, retries, and uniform error handling; maps DTOs to view models.
- **Integration**: communicates with `/auth`, `/surveys`, `/results`, `/users` endpoints over HTTPS; receives JSON.

Backend (Node.js/Express)

- **Controllers**: route handlers for auth, surveys, results, users; input validation (e.g., zod/express-validator), pagination.
- **Service Layer**: business rules (e.g., PCs can only view their groups), audit logging.
- **Auth & Sessions**: password hashing (bcrypt/argon2), secure httpOnly cookies, optional JWT for API clients, token rotation.
- **Data Access Layer**: parameterized SQL; transaction boundaries for multi-write ops.
- **Integration**: uses a pooled MySQL client; emits structured logs/metrics for ops.

Data Stores (DigitalOcean)

- **Managed MySQL**: users, groups, surveys, questions, responses, response_items; automated backups and VPC isolation.
- **Spaces/Object Storage**: export files, static assets, optional logs with lifecycle policies.

Operations (CI/CD & App Platform)

- **Build/Deploy**: on main branch, build containers, run unit tests, deploy API and static site; health checks and rollbacks.

Figure 2. Components and Data Paths

Key subsystems with their roles and how data moves from UI to database.

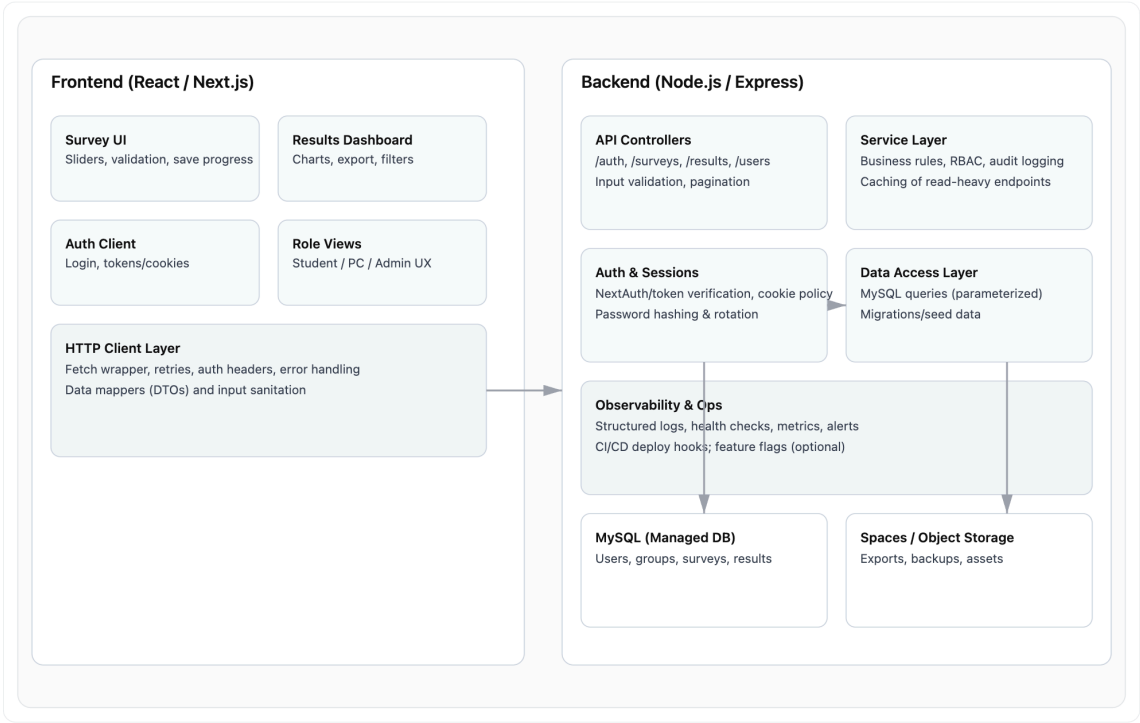


Figure 5: Components and Data Paths

Figure 4 (Mini ER Data View) shows the core entities and relationships: Users (PK `user_id`) join Groups via GroupMembers; Surveys contain SurveyQuestions; groups are linked to surveys through Assignments; Responses capture a user's submission for a survey (with denormalized `group_id` at submit time); ResponseItems store per-question answers. This schema supports role-scoped reads, longitudinal pre/post comparisons, and efficient aggregation. (Crow's foot can be added if your class prefers that notation.)

Representative API surface (abbrev).

- `POST /auth/login` → set session cookie after hash verify; returns user/roles.
- `GET /users/me` → returns profile + role; used to gate client routes.
- `GET /surveys/:surveyId` → survey + questions (role-agnostic).

- `POST /responses` → start or submit responses (supports partial save, `is_complete` on items).

- `GET /results?groupId=…` (PC/Admin) → scoped aggregate + per-student results; Students use `GET /results/me`.

- `GET /exports/group/:id` (PC/Admin) → generates CSV to Spaces (pre-signed download).

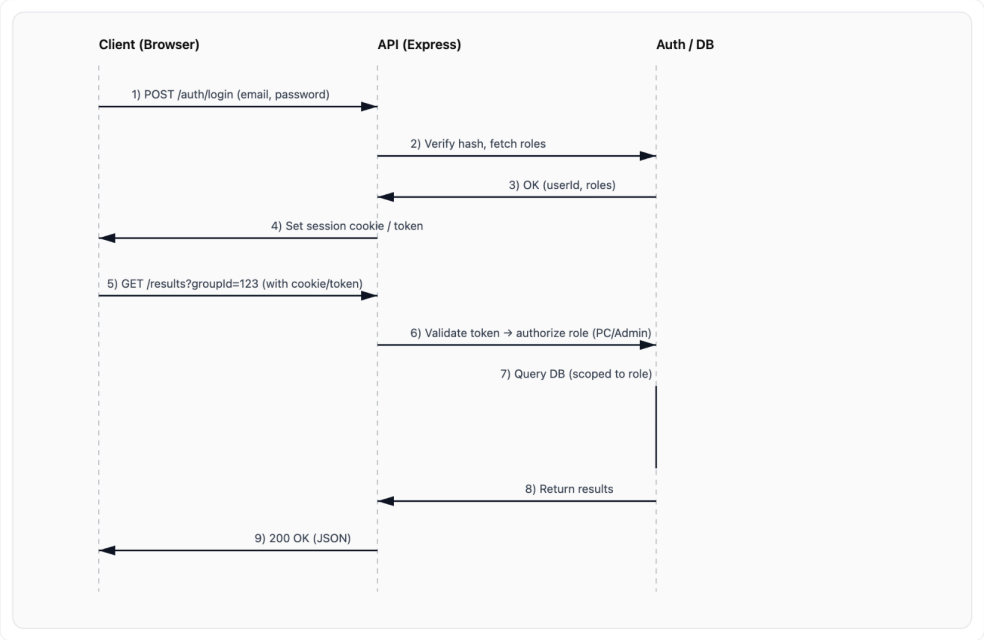
AuthN/AuthZ model.

- **Authentication**: email + password (hash stored), secure httpOnly cookie; optional short-lived JWT for programmatic access.

- **Authorization**: RBAC middleware on every protected route; Students only see their own data, PCs see assigned groups, Admins have elevated read/export scopes. See Figure 3 (Login & Session Flow).

Figure 3. Login and Session Validation Sequence

Sequence showing how the client authenticates and how the API validates role-based access.



Notes: Sessions use secure, httpOnly cookies. Role-based access ensures Students can only access their own records; Program Coordinators see only their assigned groups; Admins have elevated read scopes.

Figure 6: Validation Sequence

Validation, errors, and resilience.

- **Validation**: server-side schema validation; client mirrors constraints for better UX.

- **Errors**: uniform JSON shape `{code,message,details}`; 4xx for validation/permission, 5xx for server.

- **Resilience**: DB connection pooling; exponential backoff/retries for transient failures; idempotent POSTs where feasible.

Performance & scalability notes.

- Read-heavy endpoints (e.g., cohort dashboards) support **pagination and simple caching** at the service layer.

- **Right-sized DB instances** and query indexes (on `user_id`, `group_id`, `survey_id`) keep latency low.

- Static assets are cached; API uses gzip/br compression.

- Horizontal scaling path: add API replicas behind the App Platform; DB vertical scale as needed.

Security & privacy.

- Password hashing (argon2/bcrypt), TLS everywhere, least-privilege DB credentials, parameterized queries to prevent SQLi.

- Encrypted environment variables, role-scoped queries, audit logs for sensitive reads/exports.

- Backups tested via periodic restore drills; PII access limited to Admin/PC roles per policy.

4.3.3 Functionality

The participant interface is the most common and basic role of this application. Mainly anticipated to be students, this role's main responsibility is to take the survey and compare their results while participating in an undergraduate research program. Initially, the participant must create an account using their basic credentials (name, email, password, etc) along with an access code provided to them by the program coordinator role. The student will have joined the program and now has access to the full participant interface. They are able to view resources delegated by the program coordinator, read the about page and most importantly, take the survey (when delegated by the program coordinator). The participant will be able to take the survey and view their results immediately after completion. If the student were to take the survey multiple times throughout the program, their most recent results would be compared in the graph with their new results, allowing the participant to view their growth.

The program coordinator role is in charge of managing the surveys for respective course(es). Like the participant role, they will have to create an account using their personal credentials. They are then delegated by the administrator as a program coordinator. The program

coordinator will have a different 'card view' for each program they are responsible for. For each program, they can distribute surveys to every participant in the program, manage participant progress and view results of individual surveys. Program coordinators will also be able to provide resources to students at any point in time.

The administrator is the highest level of all 3 roles. As mentioned above, they will be able to delegate program coordinators in order to be in charge of certain courses, collect individual or entire course results and export results in order to send them to the researchers at the University of Iowa. The administrator role has the most permissions and therefore is the least common role. Although they have access to do so, their main interactions will be with program coordinators, rather than individual students themselves.

4.3.4 Areas of Challenge

Challenge 1: Cloud Migration

One of the biggest challenges we overcame was migrating from AWS cloud service to Digital Ocean. It was a challenge because none of us had much cloud experience, nor had anyone used Digital Ocean before. It was also risky since we were migrating databases with a lot of data, and we did not want to lose any of it.

In order to overcome this challenge, we took our time with the process. We did not want to rush the migration and lose any data of functionality of the application. We referenced the AWS infrastructure setup to mimic functionality on Digital Ocean. We also used outside resources in order to help us understand what services we needed to produce the required functionality of the application.

Challenge 2: Middleware Implementation

Similar to our cloud migration, middleware was challenging to implement since none of our team had any hands-on experience with it. Middleware had to be implemented everywhere within the application (frontend, backend, endpoints, etc.), so again, we did not want to compromise any features that were already working.

We were able to prevail by using outside resources (tutorial videos, various libraries, etc) in order to implement the middleware functionality. For example, we used 'bycrypt' for password hashing and implemented cookies/tokens in order for our authentication to function properly and securely.

Challenge 3: Survey exportation ideas

Our final 'big' challenge was more of a mental struggle rather than physical process difficulty. We knew we wanted to improve many aspects of the survey (format, some features, exportation, etc) since CPRE 4910 when we were in the "planning" phase with our clients. We had many different ideas on how we wanted to export the data, but never came to a distinct conclusion. As the semester of CPRE 4920 progressed, we realized this was still an aspect of the application we had not yet solved.

After some meetings with researchers of the University of Iowa, we decided to create a fourth role "researcher", whose only duty was to export all data, anonymously. This would allow the

administrator to still be able to attach results to a student, but still gave us a way to export data anonymously, per the request of the researchers we had been working with.

4.4. Technology Considerations

Front-end:

React: JavaScript library was used to create reusable components for interactive user interfaces

Pros:

- Reusable components
- Easy to integrate with other libraries
- Fast rendering

Cons:

- Steep learning curves for beginners
- Frequent updates can compromise feature compatibility

Trade-offs: Allows for fast, reusable UI development with complexity of frequent updates and steeper learning curves.

Next.js: React framework that enables server-side rendering and static site generation

Pros:

- Easy routing system
- Full-stack capabilities with API routes

Cons:

- Larger bundle size
- Steep learning curve
- Limited flexibility with custom configurations

Trade-offs: Offers enhanced performance and full-stack capabilities, but is more complex than using React by itself.

Chakra: React component library that provides customizable UI components for web applications

Pros:

- Prebuilt, responsive components
- Easy styling with consistent design system

Cons:

- Difficulties overriding when styling or complex layouts
- Limited design capabilities compared to competitors (CSS or Tailwind)

Trade-offs: Faster UI development with styled components, yet, can limit design flexibility and increase storage usage.

Front-end design solutions and alternatives: We inherited this design from two previous senior design teams. We have decided it was in our best interest to continue working with the libraries/addons they had already implemented. We have spent a lot of time in CPRE 4910 reviewing the current code and learning more about the libraries used. This allowed us to practice and ensure we understood the capabilities when the time came to implement desired features, presented by our clients.

Back-end:

Node.js: Javascript runtime that allows developers to build applications using a non-blocking, event driven architecture

Pros:

- High performance
- Scales easily
- Large ecosystem with 'npm' packages

Cons:

- Frequent updates can compromise feature compatibility
- Callback based code can be complex

Trade-offs: Provides high performance, scalability for server side development but struggles with CPU intensive jobs/processes.

Postman: Collaboration platform for API development

Pros:

- Easy API testing and debugging
- Collaboration features for teams

Cons:

- Limited (free) testing/features
- GUI dependency

Trade-offs: Makes API testing and collaboration easy, but is resource heavy.

Back-end design solutions and alternatives: Many of our team members had previous experience with at least one of the two main backend technologies implemented. We could have switched to alternatives, such as cURL or Django, however, since these technologies had already been implemented by previous teams, and we had experience with them, we decided it would be in our best interest to continue utilizing their features, already integrated with the site.

Security:

Bycrypt: Library that supports hashing and password verification

Pros:

- Easy to implement
- Hashing resistant to brute force attacks

Cons:

- Slower algorithm
- Adds computational overhead to authentication

Trade-offs: Requires careful/tedious configuration, but ensures very safe password security and provides easy hashing and salt implementation.

Tokens: Pieces of data used for authenticating/authorizing users (JWTs)

Pros:

- Carries user data securely
- Stateless authentication (no server-side needed)

Cons:

- Must be securely stored
- Revocation/expiration can be tricky to implement

Trade-offs: Require careful storage and tedious implementation but enables stateless, cross-service authentication.

Security design solutions and alternatives: Bcrypt and tokens are something new we have added to the project, since security of the website was previously lacking. We could use alternatives such as multi-factor authentication or API keys, but our cybersecurity developers decided this approach would be sufficient and improve the overall security of the application.

Cloud:

GitHub/Git: Platform used for version control and collaborative software development

Pros:

- Easy collaboration and code sharing
- Integrates with CI/CD
- Tracks changes/maintains history

Cons:

- Limited free usage
- Complex for advanced workflows

Trade-offs: Simplifies version control and collaboration but has limited free usage and can become complex for larger projects.

Digital Ocean: Cloud infrastructure provider for managing applications

Pros:

- Simple and user friendly interface
- Affordable pricing
- Fast deployment

Cons:

- Fewer advanced features than competitors (AWS)
- No team experience using this platform

Trade-offs: Offers simple, affordable cloud management solutions while lacking some advanced features and scalability.

Cloud design solutions and alternatives: One of our biggest decisions in the early stages of this project was switching cloud providers, from the previously implemented AWS. This switch was one of the first changes we started on, due to the time, complexity and learning curve increases. We would have stuck with AWS, but we believed that switching to Digital Ocean would reduce cost and decrease complexity for maintaining the site.

5. Testing

5.1. Unit Testing

- Backend service modules (e.g., Backend/api/UserNew/User.service.js (lines 20-134), Backend/api/surveyResults/surveyResults.service.js (lines 15-94), Backend/api/questions/questions.service.js (lines 16-18)) should have Jest test suites that stub the shared MySQL pool and verify each exported function: success paths, error propagation, and edge cases (missing parameters, zero results, constraint violations).
- Controllers such as Backend/api/UserNew/User.controller.js (lines 4-137) can be unit-tested with Jest + Supertest by mounting the router in an Express app and asserting status codes/payloads given mocked services and JWT secrets.
- On the frontend, utility hooks/components (iinspire-app/src/components/TheSurvey.tsx, iinspire-app/src/components/results/Profile.tsx) can be tested with React Testing Library + Jest DOM to ensure state transitions (phase changes, pagination, submission formatting) and derived text (card narratives) respond correctly to props/mock fetches.
- Where logic is mostly data transformation (e.g., grouping answers in TheSurvey), add pure helper functions and cover them with straightforward Jest assertions.

5.2. Interface Testing

- REST interfaces between frontend and backend (e.g., survey question fetch GET /api/questions, survey submission POST /api/survey-results, login POST /api/login) can be

validated with pact-style contract tests: define OpenAPI schemas or JSON Schema expectations and ensure both consumer and provider tests stay in sync. Tools: Swagger/OpenAPI for spec, Jest + Supertest + jest-openapi for server response validation, and Mock Service Worker (MSW) for client-side interface simulations.

- Composition tests should exercise flows where multiple modules interact, such as user creation (frontend form → User.controller → User.service queries) or survey submission (React state → formatting payload → surveyResults.service normalization). Write Cypress component tests or Playwright interaction scripts that mount subsets of the UI with MSW stubs to assert full request lifecycles.

5.3. Integration Testing

- Critical paths include:
    - Authentication & Authorization: POST /api/login issuing JWT tokens (requirements: secure access by role). The integration suite should spin up the Express app against a test MySQL (or dockerized DB) populated with seed data, then run Supertest flows to ensure token gate endpoints like /api/usersprogram.
    - Survey ingestion and analytics pipeline: fetching questions, filling the survey (TheSurvey) and persisting results via surveyResults.service before the dashboard (Profile) renders normalized data. Cypress/Playwright end-to-end tests can automate a student logging in, completing the survey, and verifying stored results via API assertions.
    - Program management for coordinators/admins: API endpoints for program CRUD and student listing (/api/usersprogram, /api/newprogram, /api/program-survey-results). Integration tests should verify role-based access and data joins across Users/Program/SurveyResults tables.
- Tooling: Docker Compose (MySQL + backend) for consistent environments, Jest + Supertest for API integration, Cypress or Playwright for browser-level verification against the Next.js app.

5.4. System Testing

System-level testing focused on verifying that the complete MySTEMGrowth application worked as an end-to-end system across the frontend, backend, authentication layer, and database. The strategy was to test the major workflows required by each user role and confirm that the individual units, interfaces, and integrations worked together correctly.

The system test plan was tied directly to the project requirements. Students needed to be able to create an account, log in, select a program, complete the survey, save progress, submit results, and view their results. Program Coordinators needed to be able to manage programs, view rosters, manage resources, manage tasks, configure requested demographics, and export survey results. Admins needed access to administrative dashboards, program data, access codes, and exports. Because these workflows represent the core system requirements, they were used as the basis for system testing.

The unit tests that supported system testing included backend middleware tests, service tests, and frontend utility/component tests. Middleware tests verified authentication and role-based

access control. Service tests verified logic such as duplicate email prevention, database query behavior, and expected service responses. Frontend tests verified smaller pieces such as authentication context, middleware behavior, API helpers, and component behavior where applicable.

Interface testing supported system testing by checking communication between major parts of the system. This included frontend-to-backend API requests, backend router-to-controller behavior, controller-to-service behavior, and service-to-database behavior. These interfaces were tested through a combination of Jest tests, mocked database calls, browser developer tools, backend logs, and manual API workflow testing.

Integration testing supported system testing by validating critical paths through multiple layers of the application. Critical integration paths included login and session creation, protected route access, account creation, student program selection, survey submission, survey result retrieval, Program Coordinator program management, and Admin export functionality. These paths were tested manually in the browser and verified using backend logs, database inspection, and automated tests where available.

Together, the unit tests, interface tests, and integration tests provided enough coverage for system-level confidence because they covered the core requirements and highest-risk workflows. The system was tested by running the backend server, running the frontend application, logging in as different user roles, completing role-specific workflows, checking database results, and confirming that protected pages and endpoints behaved correctly.

The main tools used for system testing were Jest, React Testing Library, browser developer tools, manual browser testing, backend console logs, and direct database inspection. Jest was used for automated unit and service testing. React Testing Library was used for frontend component behavior where applicable. Browser developer tools were used to inspect network requests, cookies, console logs, and frontend behavior. Database inspection was used to confirm that users, programs, survey responses, and results were stored correctly.

5.5. Regression Testing

We work to ensure that new additional features do not break old functionality by using standard software practice, as well as version testing and control. When implementing new features, we work off of branches, individually adding features and testing functionality. Before any of these branches are merged, we ensure that we create new, independent functions to provide any additional functionality we need to implement said new features. Each of these new functionalities are tested on their individual branches before being launched onto the main platform. In the case that functionality critical to the program made it past testing, we are able to roll back to a previous working version of the application.

Critical features to the application for students include basic functionality such as sign in, logout, ability to take the survey, and ability to view results. While the look and feel of such functions has changed slightly and may continue to be tweaked, the underlying logic has remained the same, and any future changes will undergo thorough testing before any attempt to launch new versions. Similar practices will be implemented for any changes made to Admin or Program Coordinator roles. Critical functionalities for these roles include ability to add or remove members from programs, view program information, and export program data. While updates may be made

to these functionalities, each change in functionality is rigorously monitored to ensure that these critical functions are operations at all times.

5.6. Acceptance Testing

To ensure the tool meets all functional and non-functional requirements, acceptance testing will be carried out in collaboration with our client. We demonstrate completed features in meetings, walk through the application with the client, and validate that each component behaves as intended. During these sessions, administrators, coordinators, and participant workflows will be tested, as well as distributing surveys, completing a survey, and viewing results to confirm the tool aligns with the operational needs.

5.7. Security Testing (if applicable)

Security testing was applicable because MySTEMGrowth stores and manages user accounts, program information, survey responses, survey results, access codes, and role-specific administrative functions. The main security goal was to ensure that users could only access the data and pages allowed for their role.

The protected data and resources included user profile information, passwords, authentication sessions, survey responses, survey results, program rosters, program resources, program tasks, access codes, and Admin/Program Coordinator dashboards. Because students, Program Coordinators, Researchers, and Admins have different permissions, role-based access control was a major part of the security testing strategy.

Security requirements were demonstrated through authentication, authorization, password handling, and protected route testing. The backend uses JWT-based authentication with an HTTP-only authToken cookie. Backend middleware validates the token and attaches the authenticated user to the request. Role-based middleware then checks whether the user has permission to access protected API endpoints. The frontend also uses route protection and role-based navigation to reduce accidental access to restricted pages.

Security testing included automated middleware tests using Jest. These tests checked cases such as valid tokens, missing tokens, malformed tokens, expired tokens, invalid signatures, and users with insufficient roles. This helped verify that protected backend endpoints rejected unauthorized requests and allowed valid requests from users with the correct role.

Manual security testing was also performed by logging in as different user types and attempting to access pages or actions outside each role's permissions. For example, student users should not be able to access Program Coordinator or Admin pages, and non-admin users should not be able to access admin-only endpoints. Browser developer tools were used to inspect cookies, network requests, API responses, and redirects.

Password-related security was improved by enabling hashed password login and enforcing stronger password requirements during account creation. Duplicate email prevention was also added to reduce account integrity issues. These improvements helped protect user accounts and reduce common authentication problems.

The security testing principles applied included least privilege, defense in depth, server-side authorization, protected session handling, input validation, and secure password

storage. The team also used recognized security guidance as a reference point, including OWASP ASVS, OWASP Top 10, and NIST SP 800-63B. While the project was not formally certified against these standards, they provided useful direction for authentication, access control, password handling, and web application security risks.

The primary tools used for security testing were Jest, browser developer tools, manual role-based testing, backend logs, and database inspection. Together, these methods helped demonstrate that sensitive data and role-specific functionality were protected and that the system met its main security expectations.

5.8. User Testing

User testing was completed through walkthroughs, demonstrations, and feedback from stakeholders representing the main user roles. We tested whether Students could create an account, select a program, take the survey, save progress, and view results; whether Program Coordinators could manage programs, students, resources, tasks, and exports; and whether Admins could access higher-level program and code management features. User feedback helped identify areas where the workflow needed to be clearer, such as requiring students to select a program before accessing program-specific Home, Survey, or Results pages. Overall, users responded positively to the improved navigation, loading indicators, toast messages, and expanded Program Coordinator tools. Our main observation was that the design worked best when the application clearly guided users through role-specific workflows instead of exposing every option at once.

5.9. Testing Results

Overall, testing showed that the MySTEMGrowth system is functioning well across the major required workflows. The tests covered authentication, authorization, route protection, API helper behavior, backend service logic, database query behavior, account validation, and role-based access control. Manual testing also confirmed that Students, Program Coordinators, Researchers, and Admins can use the major features intended for their roles. These results show that the project meets the main functional goals of account creation, login, program selection, survey participation, result viewing, program management, and protected access to role-specific pages and data.

Backend:

```
caleb@MacBook-Pro Backend % npm test -- --runInBand --verbose
k) (12 ms)
    ✓ authenticate returns 401 on invalid token (2 ms)
    ✓ authenticate returns 401 on expired token (3 ms)
    ✓ authenticate returns 401 when token signed with wrong secret (3 ms)
    ✓ authorizeRoles blocks users without required role
    ✓ authorizeRoles allows users with required role (1 ms)
    ✓ authorizeRoles allows when any of multiple roles match
    ✓ authorizeRoles returns 401 when req.user missing
    ✓ authorizeSelfOrRoles allows when user acts on self
    ✓ authorizeSelfOrRoles allows elevated role on another user
    ✓ authorizeSelfOrRoles blocks when neither self nor allowed role
    ✓ authorizeSelfOrRoles returns 401 when req.user missing
    ✓ authorizeSelfOrRoles handles null target id as forbidden

 PASS  tests/services/links.service.test.js
  links.service
    ✓ getAllLinks fetches by program_id
    ✓ postnewlink inserts link data
    ✓ deleteLink removes by id (1 ms)
    ✓ postNewResourceLibraryItem inserts resource library row
    ✓ getResourceLibraryByPC fetches library resources for a pc (1 ms)
    ✓ deleteResourceLibraryItem removes by resource_id and pc_id
    ✓ service methods propagate db errors (4 ms)

 PASS  tests/services/user.service.test.js
  User.service
    ✓ PostNewUser rejects an email that already has an account (1 ms)
    ✓ PostNewUser trims email before checking and inserting

 PASS  tests/services/questions.service.test.js
  questions.service
    ✓ getAllQuestions returns rows from db
    ✓ getAllQuestions propagates db errors (1 ms)

Test Suites: 4 passed, 4 total
Tests:       27 passed, 27 total
Snapshots:   0 total
Time:        0.211 s, estimated 2 s
Ran all test suites.
caleb@MacBook-Pro Backend %
```

Figure 7: Backend Testing Results

Backend testing passed completely. The backend test results showed 4 out of 4 test suites passing, with 27 out of 27 tests passing. These tests covered authentication middleware, role-based authorization, user service logic, links service logic, and questions service logic. The backend tests verified that invalid, expired, or incorrectly signed tokens are rejected, users without the required role are blocked, users with the correct role are allowed, duplicate email account creation is prevented, and service methods handle database results and errors correctly. These results provide strong evidence that the backend supports the project's security, reliability, and data-handling requirements.

Frontend:

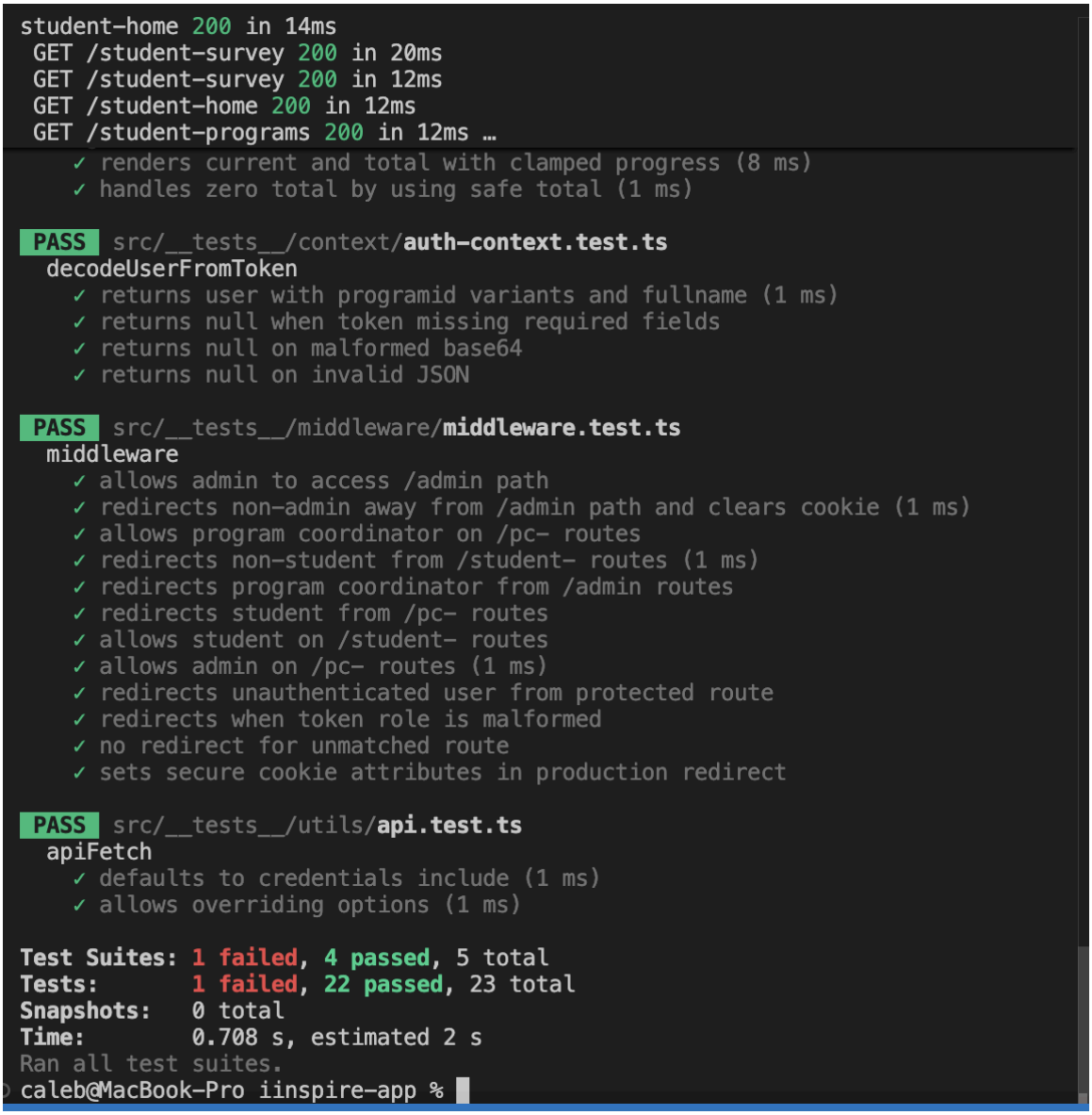


Figure 8: Frontend Testing Results

Front-end testing verified several important parts of the user interface and client-side behavior. The frontend tests covered authentication context behavior, token decoding, middleware route protection, API request configuration, and progress bar behavior. The results showed that 4 out of 5 frontend test suites passed, with 22 out of 23 tests passing. These tests confirmed that users are decoded correctly from authentication tokens, unauthorized users are redirected away from protected routes, role-based frontend access works as expected, and API requests include credentials by default. One frontend test was still failing at the time of reporting, which identifies a remaining area for cleanup, but the majority of critical frontend behavior was verified successfully.

6. Implementation

The MySTEMGrowth survey tool was a simple application that had basic functionality when our team inherited it at the beginning of senior design CPRE 4910. Since we have taken control of the project, we have innovated it into a more complete, secure and user friendly system that closely aligns to our final design. This application fully integrates frontend, backend and cloud components, along with many new security features, that all work together to provide a seamless experience for users.

Throughout our time together as a team and working with our clients in senior design, our team has successfully contributed to implementing several key features and subsystems. On the frontend side, we innovated the user interface to support both desktop and mobile devices, as well as improved the format of the survey and its display of results, as well as improved interfaces for all user types.

Backend improvements we made include improving security by implementing tokens, cookies, middleware, password hashing, endpoint protection and stronger password practices in order to ensure secure access to user data and protected routes. Additionally, we improved the cloud component by retaining functionality while migrating databases to significantly cut costs and usage.

Although I do believe that we were able to implement all the features that we initially planned, there are still some features that could be added in the future, if another senior design team gained access to the project. To start, I do think the layout of the survey can be improved. We did add all the functionality that we intended to, but the display of the survey could be better. Some pages have significantly more questions than others, and we personally think a method other than sliders might look better.

A second feature that could be added in the future is expanding the researcher's role in the application. We created this role last minute and (per the researchers at the University of Iowa), they said the only functionality they needed from that role was to be able to export data anonymously. We do believe that they could have more features, such as exporting data in various forms, comparing results from their last export or even suggesting changes to the survey. Overall, we accomplished all the goals we set for ourselves at the beginning of the project, but can see places where the application can be improved even more.

Our implementation adheres to relevant software engineering standards and best practices in areas such as security, modular design and maintainability. The use of authentication tokens and secure cookies aligns with common modern web security standards/practices for protecting user data. Our innovation allowing the website to be compatible with smaller devices supports modular design and the design of separation between components (frontend, backend, cloud) supports scalability and ease of maintenance. Overall, our final product reflects a functional, professional and innovative system that meets the primary goals of the project.

6.1. Design Analysis

The implemented design for the MySTEMGrowth tool works well and meets many of its intended goals. The application provides a clean, simple, and user-friendly interface that allows students to navigate the survey process with ease. This is supported by strong user feedback,

including an overall approval rating of 97.31%. Multiple student users noted that the website was "very simple and straightforward" and that they "liked the website as a whole," which indicates that the design effectively prioritizes usability and accessibility.

One aspect that works particularly well is the visual design and layout. Users specifically mentioned liking the color scheme, suggesting that the UI choices contributed positively to the overall experience. A consistent and visually pleasing interface likely helped reduce cognitive load and made the tool easier to engage with. The core purpose of encouraging student reflection was achieved. Feedback such as "it made me realize what areas I need to improve on" demonstrates that the survey content and structure are effective in promoting self-assessment and meaningful insights. This serves as strong evidence that the system is not only functional but also impactful.

There was also strong verbal approval from key individuals, including the client, an administrator, and a current program coordinator. Their positive feedback further validates that the system aligns well with real-world expectations and program needs. From a functional perspective, the core workflow of navigating the survey, answering questions, and reviewing results operates reliably. This indicates that the foundational design and implementation are reliable.

One of the major complications we ran into were issues with students having issues where results did not always show and surveys did not save as expected. This came from the design decision to switch the application from single program enrollment per user account to having multi-program enrollment. Because of this change there was increased complexity for the backend implication that was not properly propagated to the navbar implementation. There was a bug where if users selected the survey tab or tried to view results without selecting a program first, the backend API would not have the necessary information to retrieve the information from the database and would return an empty object. This bug is now fixed after user testing, but made it past the initial stages of development testing.

7. Ethics and Professional Responsibility

7.1. Areas of Professional Responsibility/Codes of Ethics

Area of Responsibility	**Definition**	**IEEE Code of Ethics**	**Our Team's Interaction**
Work Competence	To produce high quality work using appropriate knowledge	"To maintain and improve our technical competence"	We test our functions and work to validate questions and result score logic.
Financial Responsibility	Use project resources effectively	"To avoid wasteful practices"	We changed hosting providers to reduce cost to the fund.
Communication Honesty	Be transparent and honest in all project communication	"To be honest and realistic in stating claims or estimates"	We remain honest with our client and have full transparency as a team, to ensure that we receive honest feedback on our efforts.

Health, Safety, and Well-being	Ensure the tool promotes positive outcomes	"To hold paramount the safety, health, and welfare of the public"	We make sure our information and results encourages positive behavior
Property Ownership	Respect data ownership and rights of others	"To avoid injuring others, their property, reputation, or employment"	We ensure that the data students enter is protected, and that only people allowed to see it can see it
Sustainability	Solutions that use resources efficiently and consider long term effects on energy and maintenance	"To improve the understanding of technology; its appropriate application, and potential consequences"	We ensure our code is up to current standards with detailed documentation, as well as using minimal resources to fit the demand of the product
Social Responsibility	Designing the tool to benefit society	"To treat all persons fairly and with respect"	We prioritize inclusive language, and equitable use for students of all backgrounds

One area that our team performed well in was Financial Responsibility. One of our main contributions/improvements we made to the project was migrating clouds from AWS to Digital Ocean. We researched many alternatives and found that Digital Ocean would allow us to use all the services we would need to keep the website up and running while significantly reducing the cost it took to manage the site. Without trading any functionality, we reduced the cost from over $200 per month to $25.

One area that our team could've performed better in was Sustainability. Although we did try to improve that aspect in our project, such as switching clouds that used less services and capacity, we could've looked into more technologies/strategies to implement. We don't necessarily think we performed poorly in this area, it just wasn't one of our main areas of focus, so there is more we could've done to improve this aspect. To improve in the future, we could've referred back to this chart more frequently to ensure we were covering all areas and maximizing efforts to produce a better solution.

7.2. Four Principles

Area	**Beneficence**	**Nonmaleficence**	**Respect for Autonomy**	**Justice**
Public health, safety, and welfare	Provide support and opportunities for growth in all of the language in	Avoid harmful and discouraging language that could make students less	Make sure that students control both what information	Work to ensure all students will receive the same level of support from tool,

	the tool. Work to promote self-confidence and interest in learning.	willing to learn and grow.	they include, and how they use the information they are given from the tool.	regardless of any circumstances.
Global, cultural, and social	Offer a tool that can be used by anyone to help learn and grow in their STEM and social proficiencies.	Stay away from any culturally biased or exclusionary language throughout the tool, allowing as many people as possible to access it.	Allow programs to use this tool as they desire, leaving features that promote individuality between programs	Ensure equity by being inclusive and accessible in the tool. Make sure everyone has access to this tool.
Environmental	Use the least amount of computational resources required for a good user experience.	Avoid unnecessary runtime and data storage. Only run and store the essentials.	Allow people to delete their information or account as they desire.	Ensure environmental impact is not stuck on one program.
Economic	Make sure to use the most cost efficient hosting site to reduce cost to keep the tool running, allowing more students to use it.	Avoid requiring a paid subscription on either a program or student basis. A paid model may create income, but it will reduce the number of students that can take advantage of the tool.	Programs can choose how much they want to use the tool without having to worry about paying for a subscription they won't get enough use out of.	Allowing all programs, regardless of financial state, to use this tool allows for more students to have a better chance at a STEM related job.

Public Health, Welfare and Safety beneficence is important to our project. Our contributions to the MySTEMGrowth survey tool has made learning in undergraduate research courses related to STEM easier to measure and ensure participants are more informed about basic STEM principles. Our tool helps students visualize their growth throughout courses and gives them a visual representation of their progress. Our improvement of the UI, survey itself and results page allows students to more easily and clearly understand what aspects they are being measured on and how they relate to their growth.

Although we tried to maintain and even improve upon the Environmental aspect, there was only so much we could do to minimize the digital footprint left by our tool. We couldn't find a way

to make it function without using a cloud platform/data center, which ultimately uses electricity and water to run. However, we believe that all the beneficial innovations we've implemented outweigh the negatives that are still present. Our application promotes learning and self growth, which in turn could help students create a more sustainable method to host datacenters in the future.

7.3. Virtues

Team virtues

- Commitment: Showing up prepared and present so decisions get made and work moves forward. Weekly reports with progress and blockers, and sharing notes/action items so attendance translates into progress.
- Responsiveness: Seeking and acting on client feedback quickly to ensure fairness to stakeholders' needs. Take notes, and track follow-ups to close the loop so their voices shape the product.
- Transparency: Communicating clearly across channels (face-to-face, text, email, Discord, reports) and aligning actions with what we say.

How we support them

- Commitment: Each team member had certain responsibilities that they were expected to deliver upon. All team members regularly attended meetings, were open to help others, communicated regularly, took notes/feedback and made adjustments based on that feedback. We kept other team members and clients informed and updated with planning and progress, by demonstrating improvements in meetings as well as using gitlab issue boards, excel sheets and weekly reports to provide clarity regarding progress.
- Responsiveness: We spent a lot of our initial time communicating with our clients in order to understand what changes/innovations they wanted us to create. We wanted to grasp their concept of an ideal and complete design in order to deliver the best version of the product. We regularly presented ideas and demonstrated changes we were making as the weeks progressed. We had group discussions and took feedback from our clients in order to create the best possible final version of the application.
- Transparency: We kept consistent communication with our clients and team members on a weekly basis. Using multiple platforms (gitlab, discord, messaging and face to face in meetings), we kept all parties updated regarding any issues, ideas, and changes we were working on. We asked others for help when needed and tried to communicate to the best of our ability regarding the state of the application through both semesters of senior design (4910 & 4920).

Caleb Hemmestad:

- Virtue demonstrated: Responsibility
    - Why it's important: Responsibility is important because team projects require each member to take ownership of their work, follow through on commitments, and make sure their contributions support the overall success of the project.

    - How I've demonstrated it: I demonstrated responsibility by taking ownership of full-stack features, debugging issues across the frontend, backend, and database, and making sure changes were tested and documented.

- Virtue to strengthen: Patience
    - Why it's important: Patience is important because software development often involves unexpected bugs, unclear errors, repeated revisions, and communication challenges. Staying patient helps keep the work productive and prevents frustration from affecting decisions.
    - How I can demonstrate it: I can demonstrate patience by slowing down when debugging difficult issues, asking clarifying questions when requirements are unclear, and remaining calm when changes take multiple attempts to get right.

Ethan Buenting:

- Virtue demonstrated: Communication
    - Why it's important: Communication is a key part of any form of teamwork. Without communication you can have conflicting ideas, implementations, and accomplishing goals in parallel with each other. Communication is a huge part of applications especially because of the fact that what one person implements could rely on a teammate's code, or could be the base of future work to be done by your teammates.
    - How I've demonstrated it: Over the past year I have worked to communicate with my team frequently and thoroughly about ideas and implementations. I have updated the team on a minimum weekly basis about progress, and conferred with them frequently about ideas on how to proceed with the project.

- Virtue to strengthen: Task organization
    - Why it's important: Task organization is important because there are deadlines associated with projects, and there are many goals that need to be accomplished to meet those deadlines. While some tasks may be more important or take longer than others, it is important to consider outside factors that may cause certain tasks to be put on hold due to outside factors.
    - How I can demonstrate it: The best example of this over this semester was DNS hosting for our application. This was a fairly last minute concern due to the fact that it is not necessary to develop the application. However, this task took longer than most other goals because of the need to go through ETG for billing. Because of the outside factors involved, this task was almost not completed in time. In the future I can better plan to allot time necessary for the communication and wait time of such a task.

Ethan Van Caster:

- Virtue demonstrated: Collaboration
    - Why it's important: Collaboration helps teams work together more efficiently to solve problems and stay motivated. It allows us to complete harder tasks faster and with higher quality of work.

    - How I've demonstrated it: I worked with other members of the team on features to develop a plan and implement the features differently than how I would have on my own by leaning on their skills and experience in different aspects of the project.

- Virtue to strengthen: Time management
    - Why it's important: Clients expect deliverables in a timely manner and to do that time management is required to meet deadlines and make progress in projects.
    - How I can demonstrate it: Take advantage of issue boards in the project's gitlab by adding task breakdowns for each deliverable to plan out progress in the future to meet deadlines.

Nina Gadelha:

- Virtue demonstrated: Planning
    - Why it's important: Planning is important in order to accomplish goals and ensure everyone involved is on the same page. Jumping right into implementation can result in unexpected errors, confusion and compromization of a system.
    - How I've demonstrated it: I demonstrated planning by presenting ideas and communicating with our clients in order to ensure their input was the main focus of the project. I was continuously asking for feedback regarding plans I had made surrounding future implementation. This allowed us to create features and functionality that aligned with the clients expectations and kept us on track and in order.

- Virtue to strengthen: Collaboration
    - Why it's important: Collaboration is important because "teamwork makes the dream work". It was never feasible that a single person would be able to take on all the responsibilities and workload in order to deliver the product we made. We all contributed and created a great end product that functions well and lives up to our initial goals.
    - How I can demonstrate it: I can better demonstrate collaboration by being in regular communication with my teammates and looking to work together on features. I can use my strengths and combine them with someone else's to create a great end product, asking each other for help and advice along the way.

Ryan Mamrot:

- Virtue demonstrated: Responsibility
    - Why it's important: Responsibility is important because it ensures that tasks are completed on time and that the overall project continues to move forward. In a team environment, each member depends on others to follow through on their work, and a lack of responsibility can delay progress and impact the quality of the final product. Being responsible helps maintain accountability and trust within the team.
    - How I've demonstrated it: I demonstrated responsibility by consistently completing my assigned tasks and contributing to both the frontend development and user testing aspects of the project. I took initiative in implementing features, and ensuring that my work aligned with the overall system. I also stayed engaged

throughout the project by attending meetings, incorporating feedback, and making necessary adjustments to improve the final product.

- Virtue to strengthen: Communication
    - Why it's important: Communication is essential in a team project because it ensures that everyone is aligned on goals, progress, and expectations. Without clear communication, misunderstandings can occur, leading to inefficiencies and errors. Strong communication helps teams collaborate more effectively and produce a better final result.
    - How I can demonstrate it: I can improve my communication by being more proactive in sharing updates on my progress and asking questions when clarification is needed. I, also, can engage more frequently with my teammates to ensure alignment and provide input during discussions. By maintaining consistent communication, I can contribute to a more coordinated and efficient team environment.

Sam Craft:

- Virtue demonstrated: Independence
    - Why it's important: Being able to still get work done by yourself when you're in a team setting is critical. There are also some subprojects in a project like this that only need a single person to finish, especially if the rest of the team is busy on other issues.
    - How I've demonstrated it: I successfully added a new user role to our project based off of work done by my teammates. I also successfully updated and revised our survey to match what the University of Iowa requested.

- Virtue to strengthen: Communication
    - Why it's important: Communication is the only way to complete a group project successfully. Without communication, no one will know what is going on in each other's parts of the project.
    - How I can demonstrate it: I can work harders to communicate more frequently on both what I am working on and ask what they are working on. Even if I cannot help them directly with what they're working on, I may be able to provide advice or knowledge.

8. Conclusions

8.1. Summary of Progress

Team Accomplishments:

- Updated and refined the API to prevent unauthorized users from accessing sensitive information in our project.
- Successfully transferred our cloud infrastructure from AWS to Digital Ocean without losing any user data.

- Improved Program Coordinator dashboard to better effectively display multiple programs being managed.
- Added a Researcher role to allow researchers the ability to export anonymous data directly from the project.
- Provided support for mobile view on all pages to allow students to take the survey on any device.
- Created feature to allow students to be in multiple programs at the same time.
- Allow students to compare survey results both in the same program and across multiple programs.
- Successfully completed user testing with students currently enrolled in a program

8.2. Value Provided

For Students:

- Before:
    - Students could only be a part of one program with any given account.
    - Only the two most recent survey results would be displayed in the graph.
- Now:
    - Students are able to be a part of any programs they want with one account, and compare test results across those programs
    - Students can choose which instances of survey results will be compared on the results graph.
- Value:
    - This empowers students to learn multiple programs and years, rather than be stuck after their first program ends.
    - Students can track their growth over a longer period of time and multiple programs.

For Program Coordinators:

- Before:
    - Program Coordinators had to use a drop-down menu to select which program to view.
    - No stats about a given program were shown on the homepage.
    - Programs could not be deleted.
- Now:
    - Programs are shown in a card view, allowing multiple to be seen at once.
    - Both deleting and archiving programs have been added.
    - Program Coordinators can see at glance how many students are in each program, and how many surveys have been taken.
- Value:
    - Program Coordinators have a more efficient workflow, allowing them to expedite their work in the programs.
    - Having more features allows the Program Coordinators to use the app in a way that better suits their needs.

For Researchers:

- Before: Researchers had no access to anonymous data from the project.
- Now: Researchers have a separate user role that allows them to export all surveys with no identifying information.
- Value: This tool was built to help both students and researchers. This addition of the researcher role gives them that assistance that was not there prior.

8.3. Next Steps

While we improved this project substantially, there are still features that we did not have time to take on. This list is a combined list from the team of features that still could be implemented.

- Dynamic Code creation for Administrator, Program Coordinator and Researcher roles
- Instructional Videos in a help page for users who want additional guidance
- Scale up the backend resources as demand for the project increases

9. References

[1] M. M. Hanbury, E. L. Cox, J. M. Culley, and C. M. Pruinelli, "A web-based management application to improve data collection in an emergency care research network," *Journal of the American Medical Informatics Association*, vol. 26, no. 12, pp. 1547–1554, Dec. 2019. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC6351988/

[2] D. Lopatto, "Survey of Undergraduate Research Experiences (SURE): First findings," *CBE—Life Sciences Education*, vol. 3, no. 4, pp. 270–277, Dec. 2004. [Online]. Available: https://www.lifescied.org/doi/10.1187/cbe.04-07-0045

[3] Iowa State University, "MyStemGrowth – Senior Design Team SDMAY25-24," *Department of Electrical and Computer Engineering, Iowa State University*, 2025. [Online]. Available: https://sdmay25-24.sd.ece.iastate.edu/

[4] S. Crawford, E. R. McCabe, and L. A. Smith, "Applying web-based survey design standards," *Journal of Prevention & Intervention in the Community*, vol. 44, no. 3, pp. 181–194, 2016.

[5] D. A. Dillman, J. D. Smyth, and L. M. Christian, *Mail and Internet Surveys: The Tailored Design Method*, 4th ed. Hoboken, NJ, USA: Wiley, 2014.

[6] IEEE Computer Society, "Challenges in developing research-based web design guidelines," *IEEE Xplore Digital Library*, 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7304204.

[7] IEEE Computer Society, *IEEE Standard for Software Life Cycle Processes*, IEEE Std 12207-2017, 2017.

[8] IEEE Computer Society, *IEEE Standard for Software Configuration Management*, IEEE Std 828-2012, 2012.

[9] IEEE Computer Society, *IEEE Standard for System, Software, and Hardware Verification and Validation*, IEEE Std 1012-2016, 2016.

[10] OWASP Foundation, "OWASP Top 10 Web Application Security Risks," 2023. [Online]. Available: https://owasp.org/www-project-top-ten/

[11] DigitalOcean, "DigitalOcean App Platform Documentation," 2024. [Online]. Available: https://docs.digitalocean.com/products/app-platform/

10. Appendices

10.1. Appendix 1 – Operation Manual

The purpose of this operation manual is to provide step-by-step instructions for setting up, running, testing, demonstrating, and using the MySTEMGrowth survey system. The system consists of a frontend web application, a backend API, and a database used to store user, program, and survey result information.

1. System Requirements
Before running the system, make sure the following software is installed:
• Node.js
• npm
• Git
• MySQL
• A modern web browser such as Google Chrome, Microsoft Edge, or Safari
• A code editor such as Visual Studio Code

The project is divided into two main application folders:
• iinspire-app: the frontend Next.js application
• Backend: the backend Express API

The frontend is responsible for displaying the user interface, collecting survey responses, showing dashboards, and presenting survey results. The backend is responsible for authentication, authorization, database communication, survey result processing, and API endpoints.

2. Environment Setup
First, open the project repository on the local machine. If the repository has not already been cloned, clone it using Git and then open the project folder.

Install the frontend dependencies:
cd iinspire-app
npm install

Install the backend dependencies:
cd Backend
npm install

Next, configure the environment variables required by the frontend and backend.

The frontend should include an environment variable that points to the backend API URL. For local development, this will usually point to the backend server running on port 5001.

Example frontend environment variable:
NEXT_PUBLIC_API_URL=http://localhost:5001

The backend should include environment variables for the database connection, authentication secret, and server configuration.

Example backend environment variables:
PORT=5001
DB_HOST=localhost
DB_USER=your_database_user
DB_PASSWORD=your_database_password
DB_NAME=your_database_name
JWT_SECRET=your_jwt_secret
NODE_ENV=development

The exact values may differ depending on the development machine, database setup, or cloud deployment configuration.

3. Running the System
Start the backend server first:
cd Backend
npm run dev

The backend should start on the configured port, usually:
http://localhost:5001

After the backend is running, start the frontend application:
cd iinspire-app
npm run dev

The frontend should start at:
http://localhost:3000

Open the frontend URL in a web browser. The login page should appear.

4. Basic User Roles
The system supports multiple user roles:
• Student
• Program Coordinator
• Researcher
• Admin

Each role has access to different parts of the system. Students can create an account, join programs, take surveys, save progress, and view their results. Program Coordinators can manage programs, view students, manage resources, manage tasks, select requested demographics, and

export survey results. Admins can view administrative dashboards, access program information, manage access codes, and download results. Researchers may be used for viewing or analyzing information depending on the configured permissions.

5. Account Creation and Login
To create a new student account:
1. Open the application in the browser.
2. Click the create account link from the login page.
3. Enter an email address.
4. Enter the access code provided for the appropriate role or program.
5. Enter the user's full name.
6. Enter a valid password.
7. Click the Create Account button.

The system validates password requirements before submitting the request. If the account is created successfully, the user is returned to the login screen. If the email address already has an account, the system prevents duplicate account creation and displays an error message.

To log in:
1. Open the login page.
2. Enter the user's email address.
3. Enter the user's password.
4. Click Login.
5. The system redirects the user to the appropriate dashboard based on their role.

6. Student Workflow
After logging in as a student, the user is taken to the student program dashboard.

To use the student workflow:
1. Log in as a student.
2. On the student dashboard, select a program.
3. After selecting a program, the student can access the program home page.
4. From within a selected program, the student can open the survey.
5. Complete the survey questions.
6. Submit the survey when all required questions are complete.
7. View survey results from the results page.

The student must select a program before accessing program-specific pages such as the survey or results page. This ensures that survey results are associated with the correct user and program. If survey progress saving is enabled, students can partially complete a survey and return later without losing progress.

7. Program Coordinator Workflow
After logging in as a Program Coordinator, the user is taken to the Program Coordinator dashboard.

To use the Program Coordinator workflow:
1. Log in as a Program Coordinator.
2. Open the Program Coordinator dashboard.
3. Select an existing program or create a new program.
4. View the program dashboard.
5. Use the roster page to view enrolled students.

6. Use the resources page to add, remove, or manage student resources.
7. Use the tasks page to create or manage program tasks.
8. Configure demographic fields requested from students.
9. Export survey results when needed.

Program Coordinators should select a program before using program-specific tools. This ensures that resources, tasks, students, and survey results are connected to the correct program.

8. Admin Workflow
After logging in as an Admin, the user is taken to the Admin dashboard.

To use the Admin workflow:
1. Log in as an Admin.
2. Open the Admin dashboard.
3. View the list of programs.
4. Generate or view access codes for Admin or Program Coordinator accounts.
5. Download survey results for programs.
6. Review program submission information.
7. Manage administrative functions as needed.

Admins have broader access than other user roles and are responsible for higher-level system oversight.

9. Demonstration Instructions
A standard system demonstration can follow this sequence:
1. Start the backend server.
2. Start the frontend application.
3. Open the application in a browser.
4. Demonstrate the login page.
5. Log in as an Admin.
6. Show the Admin dashboard and access code section.
7. Log out.
8. Log in as a Program Coordinator.
9. Select or create a program.
10. Show the program dashboard, roster, resources, tasks, and export features.
11. Log out.
12. Log in as a Student.
13. Select a program from the student dashboard.
14. Open the survey.
15. Complete or partially complete the survey.
16. Submit the survey.
17. Show the student results graph.
18. Demonstrate that the results are tied to the selected program.

This flow demonstrates the major user roles and the connection between frontend interaction, backend processing, and stored survey data.

10. Testing Instructions
To run frontend tests:
cd iinspire-app
npm test

To run backend tests:
cd Backend
npm test

Manual testing should also be performed to verify the full application workflow.

Manual testing checklist:
• Valid users can log in.
• Invalid login attempts show an error.
• Users are redirected based on their role.
• Sign out clears the user session.
• Account creation works for valid inputs.
• Duplicate email account creation is prevented.
• Password requirements are enforced.
• Students can select a program.
• Students cannot access program-specific survey pages without selecting a program.
• Students can complete and submit a survey.
• Survey progress can be saved during partial completion.
• Survey results appear for the correct user and program.
• Program Coordinators can view students in a program.
• Program Coordinators can manage resources.
• Program Coordinators can manage tasks.
• Admin users can access the Admin dashboard.
• Unauthorized users are blocked from protected pages.
• Export and download features work correctly.

11. Troubleshooting
If the backend does not start:
• Confirm that backend dependencies are installed.
• Check the backend .env file.
• Confirm that MySQL is running.
• Confirm that the database credentials are correct.
• Confirm that the backend port is not already in use.

If the frontend does not start:
• Confirm that frontend dependencies are installed.
• Check that the correct Node.js version is installed.
• Confirm that the frontend environment variables are set.
• Restart the development server.

If the frontend cannot connect to the backend:
• Confirm that the backend server is running.
• Check NEXT_PUBLIC_API_URL.
• Confirm that CORS is configured correctly on the backend.
• Confirm that requests are being sent to the correct backend port.

If login or session behavior is not working:
• Clear browser cookies.
• Log out and log back in.
• Restart both frontend and backend servers.

• Confirm that the backend JWT secret is configured.
• Confirm that authentication cookies are being set.

If survey results do not appear:
• Confirm that the student selected the correct program.
• Confirm that survey results exist for the correct user ID and program ID.
• Confirm that the survey was submitted successfully.
• Check the backend logs for API errors.

If role-based access is not working:
• Confirm that the user has the correct role in the database.
• Confirm that the authentication token includes the correct role.
• Check frontend middleware and backend authorization middleware.
• Log out and log back in after changing a user role.

12. Shutdown Instructions
To stop the application during local development:
1. Stop the frontend server by pressing Ctrl + C in the frontend terminal.
2. Stop the backend server by pressing Ctrl + C in the backend terminal.
3. Stop the database service if it was started manually.

13. Summary
The MySTEMGrowth system is operated by running the backend API, running the frontend application, and connecting both to the database. Users interact with the frontend through role-specific pages, while the backend handles authentication, authorization, survey processing, and database communication. The system supports students, Program Coordinators, Researchers, and Admins, with each role receiving access to the features needed for their responsibilities.

10.2. Appendix 2 – Alternative/initial version of design

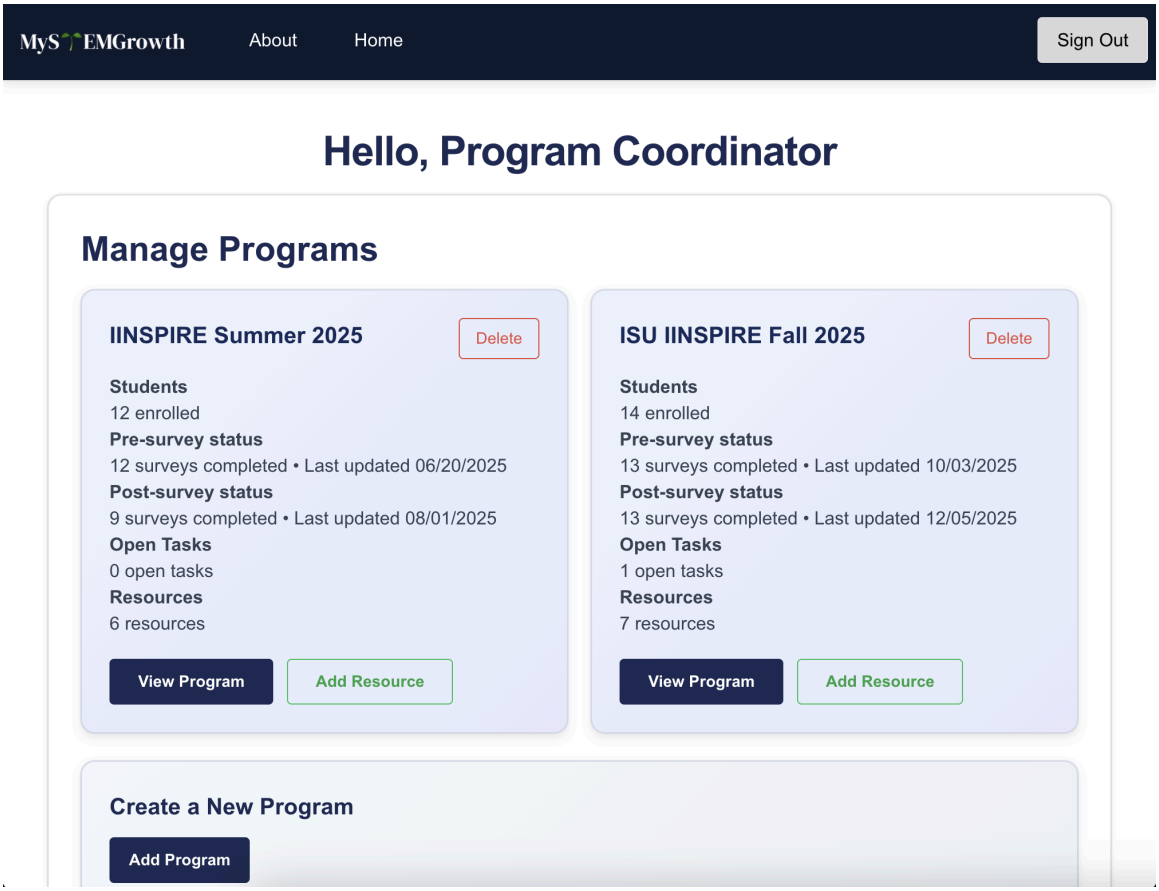


Figure 9: Version 1

This version was modified because the client didn't like the empty space at the bottom for one button.

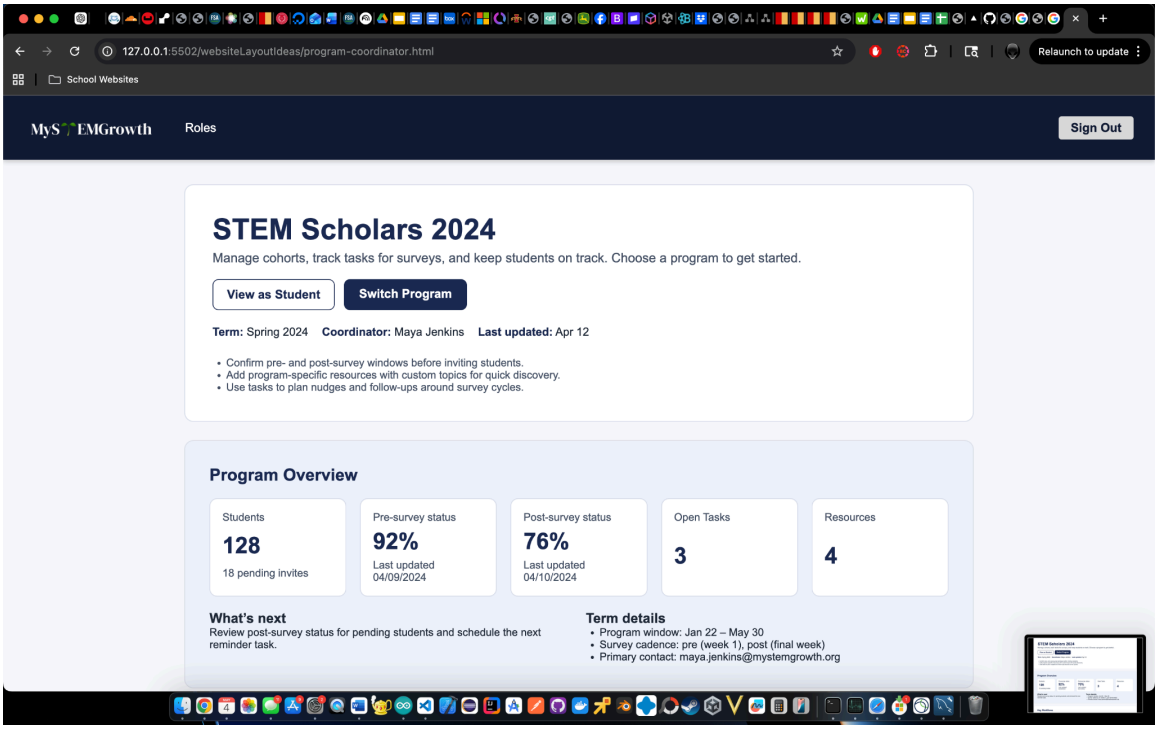


Figure 10: Version 2

This version was from early prototyping so some things from here were not actually included in the actual application.

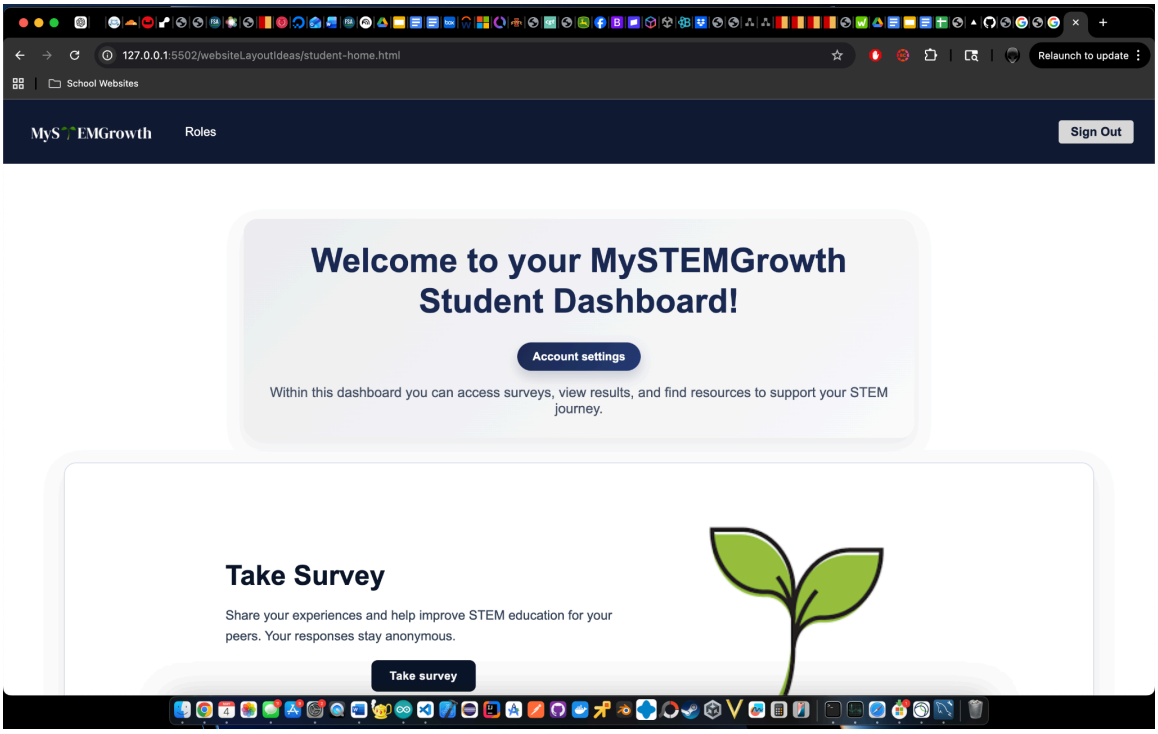


Figure 11: Version 3

This is another version from early prototyping. In the actual app we ended up putting the settings in the navbar so that all users can easily make changes without needing a whole new page for every user.

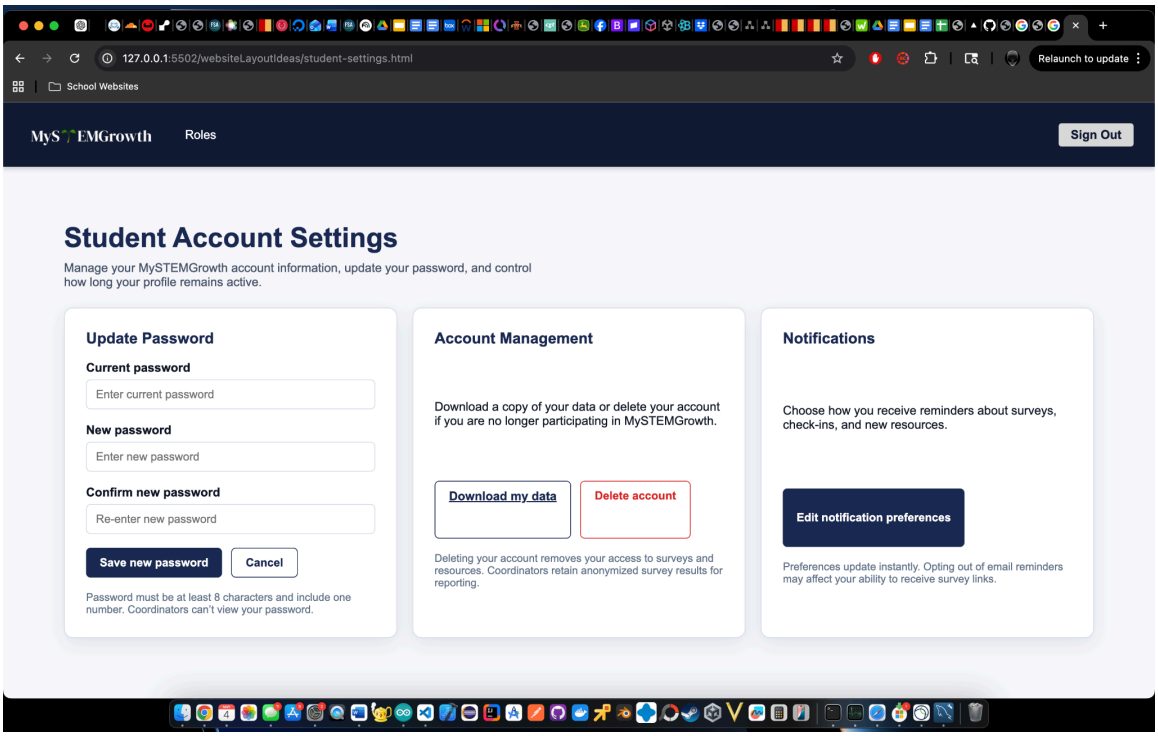


Figure 12: Version 4

This is an early prototyping version of the settings which we modified to allow users to do more than just change their password and also add demographic information for when the Program Coordinator requests it.

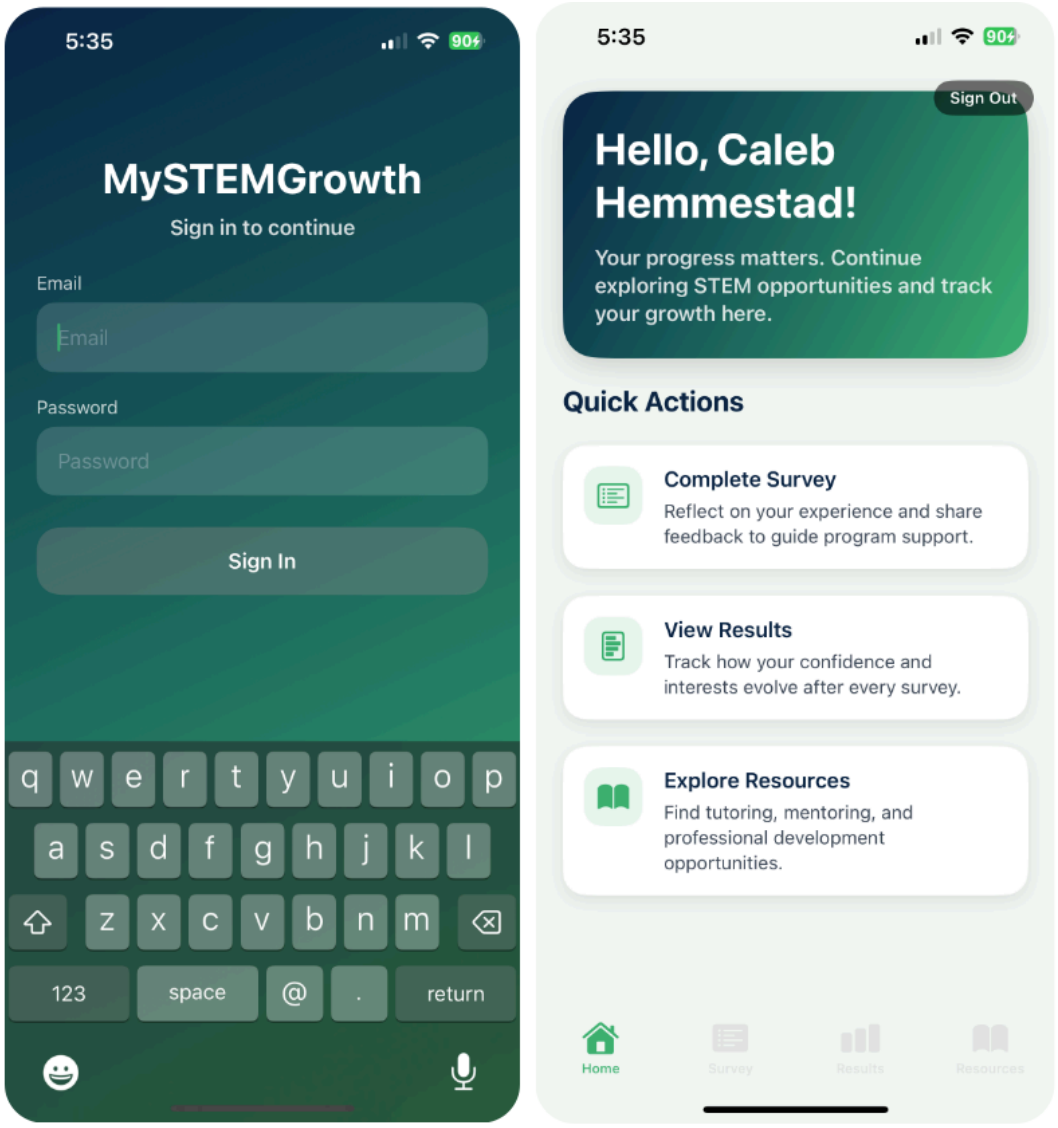


Figure 13: Mobile Idea

We ended up ditching the mobile app idea because it would create a new codebase that someone would have to be able to maintain.

10.3. Appendix 3 – Other considerations

We all learned a lot from working on this project for 2 semesters. One of the most important skills we learned was teamwork. Through this entire project, we all had to agree upon a common goal, split the workload, ask for help and trust each other to deliver upon their promises. Although we had participated in group projects before, we had never worked this long with the same team. Due to this, we developed trust, comfortability and friendship. Week by week, it was less nerve racking to enter a meeting and by the end, it just felt like you were spending time with a group of friends.

We also learned new skills that we had never tried before. The cloud migration and security implementation were complex skills that we had never conducted before. Through research, trial and error and teamwork, we were able to implement everything we set out to at the beginning of CPRE 4910.

Finally, we learned and demonstrated responsibility. Not only did we have our teammates relying on us, but we also had our clients. This was a new experience for everyone, having a "boss" that we had to plan with. We had to ensure we agreed upon a vision and a plan, in order to execute a final project that they expected and liked.

Overall, not only did we all learn a lot of new technical skills and soft skills but we also had a lot of fun. Collaborating with each other, brainstorming ideas and putting our own little twist on some of the features was very entertaining and new. We all enjoyed our time in senior design I and II. Although we will miss it, we are excited to take these new experiences and skills and apply them to whatever is next in life!

10.4. Appendix 4 – Code

https://git.ece.iastate.edu/sd/sdmay26-27

```
const decodeRoleFromToken = (token?: string | null): string | null => {
  if (!token) return null;
  try {
    const base64Url = token.split('.')[1];
    if (!base64Url) return null;
    const base64 = base64Url.replace(/-/g, '+').replace(/_/g, '/');
    const decode =
      typeof atob === 'function'
        ? (b64: string) => atob(b64)
        : (b64: string) => Buffer.from(b64, 'base64').toString('utf8');
    const jsonPayload = decode(base64);
    const payload = JSON.parse(jsonPayload);
    return payload.role ? String(payload.role).toLowerCase() : null;
  } catch {
    return null;
  }
};
```

Figure 14: Code Snippet 1

```
const express = require('express');
const programController = require('./program.controller')
const { authenticate, authorizeRoles } = require('../../middleware/auth');
const router = express.Router();

//define routes and map them to controller methods
router.post("/newprogram", authenticate, authorizeRoles('Admin', 'ProgramCoordinator'),
router.put("/editprogram", authenticate, authorizeRoles('Admin', 'ProgramCoordinator'),
router.get("/getprograms", authenticate, authorizeRoles('Admin'), programController.get

router.post("/getPCprograms", authenticate, authorizeRoles('ProgramCoordinator', 'Admi

router.delete("/deleteProgram", authenticate, authorizeRoles('Admin', 'ProgramCoordinat
router.put("/archiveProgram", authenticate, authorizeRoles('Admin', 'ProgramCoordinato
router.put("/unarchiveProgram", authenticate, authorizeRoles('Admin', 'ProgramCoordinat

module.exports = router;
```

Figure 15: Code Snippet 2

```
import type { ChangeEvent } from "react";
import { Suspense, useCallback, useEffect, useState } from "react";
import { Box, Button, Flex, Heading, HStack, SimpleGrid, Stack, Text, Wrap, WrapItem }
import { useRouter, useSearchParams } from "next/navigation";
import BackFooter from "@/components/util/back-footer";
import Footer from "@/components/util/footer";
import Navbar from "@/components/util/navbar";
import { useAuth } from "@/context/auth-context";
import colors from "../../../public/colors";
import { formatDateMMDDYYYY } from "@/utils/date";
import { AddResourceToProgramDialog } from "./modals/AddResourceToProgramDialog";
import { AddPendingTaskDialog } from "./modals/AddPendingTaskDialog";
import { ChooseInviteEmailDialog } from "./modals/ChooseInviteEmailDialog";
import { RemoveResourceFromProgramDialog } from "./modals/RemoveResourceFromProgramDial
import { RemoveStudentFromProgramDialog } from "./modals/RemoveStudentFromProgramDialog
import { PCBlueButton } from "./util/blueButton";
import { PCGreenButton } from "./util/greenButton";
import { showErrorToast, showSuccessToast } from "./util/notifications";
import { fetchProgramsByCoordinator, updateProgramDemographics } from "./util/programAp
import { PCRedButton } from "./util/redButton";
import {
  addLibraryResourceToProgram,
  fetchAvailableLibraryResourcesForProgram,
  fetchProgramResources,
  removeProgramResourceByLinkId,
} from "./util/resourceApi";
import { fetchProgramStudents, removeStudentFromProgram } from "./util/studentApi";
```

Figure 16: Code Snippet 3

10.5. Appendix 5 – Team Contract

Complete each section as completely and concisely as possible. We strongly recommend using tables or bulleted lists when applicable.

Team Members

Caleb Hemmestad: Backend/Cloud

Ethan Buenting: Backend/Cloud

Ethan Van Caster: Frontend Developer

Nina Gadelha: Cybersecurity Developer

Ryan Mamrot: Frontend Developer

Sam Craft: Backend Developer

Required Skill Sets for Your Project

- Frontend: React/Next.js (TypeScript), Java, Chakra UI/Tailwind, accessibility basics, Jest/RTL testing, GitHub workflows.
- Backend: Node.js/Express, REST API design, MySQL/SQL, Jest/Supertest testing, GitHub workflows.
- Cloud/DevOps: CI/CD and releases, Docker basics, environment/secrets management, deployment to hosting platforms.
- Security: Password hashing (bcrypt), JWT/token auth, secure cookies (httpOnly/secure/sameSite), input validation/OWASP awareness, CSRF/HTTPS hardening, dependency patching/CVE response, vulnerability testing.

Skill Sets Covered by the Team

Caleb Hemmestad: JS/TS, Next.js, React, Express/Node, REST API, JWT, Cookies, Middleware, MySQL/SQL, Jest, Docker, PlantUML/Mermaid, Documentation, Prototyping, GitLab

Ethan Buenting: Next.js, Node.js, JavaScript, Java, SQL, AWS, GitLab, Digital Ocean

Ethan Van Caster: React, Next.js, Java, JS, Chakra UI, HTML, CSS, SQL, environment management, GitLab

Nina Gadelha: Object-oriented programming, password hashing, vulnerability testing, React, Java, Typescript, HTML/CSS, github

Ryan Mamrot: Web/app development, frontend/backend development, database, Data Analysis, and virtual machine experience

Sam Craft: Object-oriented programming, Node.js, Express, SQL/MySQL, Gitlab, Docker

Project Management Style Adopted by the team

Our team has decided to implement the Agile development process during Senior Design I (CPRE 4910). Throughout this semester in Senior Design I, we believe we have discovered many adjustments/innovations our team is able to make to the website, in order to satisfy our clients and future users. Working on features in a 'cyclical' process ensures our improvements are continuously tested. We want to ensure all changes we make benefit the overall state of the website, work as expected for all types of users and do not compromise any other feature functionality. Adopting the Agile process allows us to frequently check our features, while integrating innovations frequently.

Individual Project Management Roles

Cloud transition + backend management: Ethan Buenting and Caleb Hemmestad

Mobile screen configurations + team contact: Ethan Van Caster

Security implementations + misc. features: Nina Gadelha and Caleb Hemmestad

Frontend + survey development: Ryan Mamrot

Backend development + testing: Sam Craft

Team Contract

Team Members:

1) Nina Gadelha
2) Caleb Hemmestad
3) Ethan Buenting
4) Ethan Van Caster
5) Ryan Mamrot
6) Sam Craft

Team Procedures

1. Team meetings will be held primarily face-to-face in whatever rooms we can reserve in the SIC or the TLA. Occasionally due to schedule conflicts we will hold impromptu meetings virtually or communicate over our Discord channel. The meeting times we set aside were:
    a. Monday 4-5 pm
    b. Tuesday 4:30 - 5:30 pm
    c. Thursday 2 - 3 pm
2. Texting, Discord, Email, Face-to-face meetings, and lecture are all valid methods of communication for the group
3. We will use majority vote as our decision making policy
4. We will create a Google Drive folder for all shared documents, such as class documents. There will be a minutes folder created for all meetings in case a member of the group is unable to participate. We will also have a shared Git repository for all relevant code.

Participation Expectations

1. Attendance, punctuality, and participation are expected in all meetings. If someone must miss a meeting, at least an hour notice is expected. From there, the team can either reschedule the meeting or continue as scheduled. The person who missed the meeting should reach out and get the information they missed.

2. All team members are expected to contribute to all assignments and portions of the projects that are relevant to their major. While certain assignments or tasks may be taken on by individuals who feel it is within their expertise, it is expected that overall this project is an equal group effort.

3. Regular communication is expected regarding project progress/problems, meeting attendance and communication from advisors/clients.

4. Each team member has a say when making decisions. Depending on the role/experience/expertise of a certain member, their input may weigh more (dependent). On average, every member will get an equal say.

Leadership

1. The roles of the group are dedicated as follows:

    a. Caleb: Cybersecurity Development + Backend

    b. Nina: Cybersecurity Development

    c. Sam: Backend + Testing

    d. Ethan: Team Organizer + Cloud

    e. VC: Frontend + Client Contact

    f. Ryan: Frontend Development

2. We will use Gitlab's Tasks function in order to monitor progress and member contributions as well as delegate work. During team meetings we can work on adding Tasks for features we want to begin implementing, as well as helping us to plan for the future.

3. Members will receive the praise of the team for their contributions, as well as the satisfaction of a successful project.

Collaboration

1. Individual skills and expertise that individual members bring to the team:

    a. Caleb: Full-Stack Website Development with a Cybersecurity mindset and Cloudflare/AWS experience.

b. Nina: Frontend App development, some backend experience, VM experience, web development, linux proficient

c. Sam: Web Development, Frontend/Backend, Database, DigitalOcean, AWS

d. Ethan: Frontend/Backend, MySQL and PostgreSQL, Render

e. VC: Previous web development, databases, data analysis

f. Ryan: Web/App development, Frontend/Backend, database, Data Analysis, and some VM experience

2. Individuals are encouraged to share their ideas and thoughts on project features or potential implementations at team meetings, where the team can vote on them and/or make changes/improvements to the proposition that the group agrees on with the majority.

3. If any member feels that the team environment or an individual is obstructing their ability or opportunity to contribute, we would expect that they bring the conflict up at a team meeting so it can be resolved swiftly and amicably. If the individual does not feel comfortable presenting the issue in a group discussion, they can bring it to the attention of the group coordinator or an instructor of the course.

Goal-Setting, Planning, and Execution

1. Our team goals for this semester are to create a ready-to-launch deliverable, have fun, learn new skills, create new friendships, and build new connections amongst our team.

2. We plan to divide work among members heavily based on their skill sets to allow them to build within their respective areas, as well as work as a team to create a clear and decisive list of tasks and goals to help us work concisely as a team.

3. To ensure an efficient execution, we plan on keeping the team mutually accountable between group members both within team meetings and weekly progress reports.

Consequences for Not Adhering to Team Contract

1. We will handle infractions of any obligations by individual members within team meetings by having a group conversation with that member to determine reasoning and developing an actionable plan to fix the issue.

2. If infractions continue past the previously mentioned expectations and attempted solutions, we will contact the course instructor(s), Dr. Fila and Dr. Rover, to help with an outside opinion to provide us with a potential solution or course of action.

****

a. I participated in formulating the standards, roles, and procedures as stated in this contract.
b. I understand that I am obligated to abide by these terms and conditions.
c. I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.

1) Nina Gadelha ` DATE 9/4/2025

2) Caleb Hemmestad DATE 9/5/2025

3) Sam Craft DATE 9/8/2025

4) Ethan Van Caster DATE 9/8/2025

5) Ethan Buenting DATE 9/8/2025

6) Ryan Mamrot DATE 9/9/2025